GeoServer User Manual

Release 2.15.1

GeoServer

April 29, 2019

Contents

GeoServer is an open source software server written in Java that allows users to share and edit geospatial data. Designed for interoperability, it publishes data from any major spatial data source using open standards.

This User Manual is a comprehensive guide to all aspects of using GeoServer. Whether you are a novice or a veteran user of this software, we hope that this documentation will be a helpful reference.

Introduction Learn about GeoServer.

Installation Install GeoServer for your platform.

Getting started Tutorials to help you get started with GeoServer.

Web administration interface Navigate the GeoServer graphical interface.

Data management Load and publish data from a variety of sources.

Styling Styling and visualization for published layers in GeoServer.

Services OGC services, the primary method of publishing data in GeoServer.

Filtering Filter your requests and responses to increase efficiency.

Server configuration Configuration options for GeoServer administrators.

GeoServer data directory Settings and configuration files for GeoServer.

Running in a production environment Best practices for using GeoServer.

REST Interact programmatically with GeoServer without using the graphical interface.

Security Secure and restrict access to data and services.

GeoWebCache Accelerate your GeoServer with tile caching.

Extensions Add extra functionality to GeoServer with extensions.

Community modules Add cutting-edge functionality to GeoServer with community modules.

Tutorials Other tips and tricks for getting the most out of your GeoServer instance.

CHAPTER 1

Introduction

This section gives an overview of GeoServer the project, its background, and what it can do for you.

For those who wish to get started with GeoServer right away, feel free to skip to the *Installation* section.

1.1 Overview

GeoServer is an open source software server written in Java that allows users to share and edit geospatial data. Designed for interoperability, it publishes data from any major spatial data source using open standards.

Being a community-driven project, GeoServer is developed, tested, and supported by a diverse group of individuals and organizations from around the world.

GeoServer is the reference implementation of the Open Geospatial Consortium (OGC) Web Feature Service (WFS) and Web Coverage Service (WCS) standards, as well as a high performance certified compliant Web Map Service (WMS). GeoServer forms a core component of the Geospatial Web.

1.2 History

GeoServer was started in 2001 by The Open Planning Project (TOPP), a non-profit technology incubator based in New York. TOPP was creating a suite of tools to enable open democracy and to help make government more transparent. The first of these was GeoServer, which came out of a recognition that a suite of tools to enable citizen involvement in government and urban planning would be greatly enhanced by the ability to share spatial data.

The GeoServer founders envisioned a Geospatial Web, analogous to the World Wide Web. With the World Wide Web, one can search for and download text. With the Geospatial Web, one can search for and download spatial data. Data providers would be able to publish their data straight to this web, and users could directly access it, as opposed to the now indirect and cumbersome methods of sharing data that exist today.

Those involved with GeoServer founded the GeoTools project, an open source GIS Java toolkit. Through GeoTools, support for shapefiles, Oracle databases, ArcSDE integration, and much more was added.

Around the same time as GeoServer was founded, The OpenGIS Consortium (now the Open Geospatial Consortium) was working on the Web Feature Service standard. It specifies a protocol to make spatial data directly available on the web, using GML (Geographic Markup Language), an interoperable data format. A Web Map Service was also created, a protocol for creating and displaying map images created from spatial data.

Other projects became interrelated. Refractions Research created PostGIS, a free and open spatial database, which enabled GeoServer to connect to a free database. Also, MetaCarta originally created OpenLayers, an open source browser-based map viewing utility. Together, these tools have all enhanced the functionality of GeoServer.

GeoServer can now read data from over a dozen spatial data sources and output to many different formats. Now in its second decade, GeoServer is continuing on its mission to make spatial data more accessible to all.

1.3 Getting involved

GeoServer exists because of the efforts of people like you.

There are many ways that you can help out with the GeoServer project. GeoServer fully embraces an open source development model that does not see a split between user and developer, producer and consumer, but instead sees everyone as a valuable contributor.

1.3.1 Development

Helping to develop GeoServer is the obvious way to help out. Developers usually start with bug fixes and other small patches, and then move into larger contributions as they learn the system. Our developers are more than happy to help out as you learn and get acquainted. We try our hardest to keep our code clean and well documented.

You can find the project on GitHub. As part of the GitHub model, anyone can submit patches as pull requests, which will be evaluated by the team. To learn more about contributing to the GeoServer codebase, we highly recommend joining the GeoServer developers mailing list. See details below.

1.3.2 Documentation

Another crucial way to help out is with documentation. Whether it's adding tutorials or just correcting mistakes, every contribution serves to make the project more healthy. And the best part is that you do not need to be a developer in order to contribute.

Our official documentation is contained as part of our official code repository. As part of the GitHub model, anyone can submit patches as pull requests, which will be evaluated by the team.

To learn more about contributing to the GeoServer codebase, we highly recommend joining the GeoServer developers mailing list (see details below). For typos and other small changes, please see our Documentation Guide for how to make quick fixes.

1.3.3 Mailing lists

GeoServer maintains two email lists:

- GeoServer Users
- GeoServer Developers

The Users list is mainly for those who have questions relating to the use of GeoServer, and the Developers list is for more code-specific and roadmap-based discussions. If you see a question asked on these lists that you know the answer to, please respond!

These lists are publicly available and are a great resource for those who are new to GeoServer, who need a question answered, or who are interested in contributing code.

1.3.4 IRC

Users can join the IRC channel, #geoserver, on the Freenode network, in order to collaborate in real time. GeoServer developers occasionally will be in this channel as well.

1.3.5 Bug tracking

If you have a problem when working with GeoServer, then please let us know through the mailing lists. GeoServer uses JIRA , a bug tracking website, to manage issue reports. In order to submit an issue, you'll need to create an account first.

Everyone is encouraged to submit patches and, if possible, fix issues as well. We welcome patches through JIRA, or pull requests to GitHub.

Responsible Disclosure

Warning: If you encounter a security vulnerability in GeoServer please take care to report the issue in a responsible fashion:

- Keep exploit details out of issue report (send to developer/PSC privately – just like you would do for sensitive sample data)
- Mark the issue as a vulnerability.
- Be prepared to work with Project Steering Committee (PSC) members on a solution

Keep in mind PSC members are volunteers and an extensive fix may require fundraising / resources

If you are not in position to communicate in public please consider commercial support, contacting a PSC member, or reaching us via the Open Source Geospatial Foundation at info@osgeo.org.

1.3.6 Translation

We would like GeoServer available in as many languages as possible. The two areas of GeoServer to translate are the text that appears in the *Web administration interface* and this documentation. If you are interested in helping with this task, please let us know via the mailing lists.

1.3.7 Suggest improvements

If you have suggestions as to how we can make GeoServer better, we would love to hear them. You can contact us through the mailing lists or submit a feature request through JIRA.

1.3.8 Spread the word

A further way to help out the GeoServer project is to spread the word. Word-of-mouth information sharing is more powerful than any marketing, and the more people who use our software, the better it will become.

1.3.9 Fund improvements

A final way to help out is to push for GeoServer to be used in your own organization. A number of commercial organizations offer support for GeoServer, and any improvements made due to that funding will benefit the entire GeoServer community.

1.4 License

For complete license details review `LICENSE.txt`.

GeoServer is free software and is licensed under the `GNU General Public License`:

```
GeoServer, open geospatial information server
Copyright (C) 2014-2016 Open Source Geospatial Foundation.
Copyright (C) 2001-2014 OpenPlans

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 2 of the License, or
(at your option) any later version (collectively, "GPL").

As an exception to the terms of the GPL, you may copy, modify,
propagate, and distribute a work formed by combining GeoServer with the
EMF, XSD and OSHI Libraries, or a work derivative of such a combination, even if
such copying, modification, propagation, or distribution would otherwise
violate the terms of the GPL. Nothing in this exception exempts you from
complying with the GPL in all respects for all of the code used other
than the EMF, XSD and OSHI Libraries. You may include this exception and its grant
of permissions when you distribute GeoServer.  Inclusion of this notice
with such a distribution constitutes a grant of such permissions.  If
you do not wish to grant these permissions, remove this paragraph from
your distribution. "GeoServer" means the GeoServer software licensed
under version 2 or any later version of the GPL, or a work based on such
software and licensed under the GPL. "EMF, XSD and OSHI Libraries" means
Eclipse Modeling Framework Project and XML Schema Definition software
distributed by the Eclipse Foundation and the OSHI library, all licensed
under the Eclipse Public License Version 1.0 ("EPL"), or a work based on
such software and licensed under the EPL.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 51 Franklin Street, Suite 500, Boston, MA 02110-1335  USA
```

This product includes software developed by the Apache Software Foundation (http://www.apache.org/) licensed under the `Apache License Version 2.0` and `Apache License Version 1.1`.

This product includes software developed by the Eclipse Software Foundation under the `Eclipse Public License`.

CHAPTER 2

Installation

There are many ways to install GeoServer on your system. This section will discuss the various installation paths available.

If using Windows or OS X, we recommend using the installers.

Note: To run GeoServer as part of an existing servlet container such as Tomcat, please see the *Web archive* section.

Warning: GeoServer requires a Java 8 environment (JRE) to be installed on your system. This must be done prior to installation.

2.1 Windows installer

The Windows installer provides an easy way to set up GeoServer on your system, as it requires no configuration files to be edited or command line settings.

1. Make sure you have a Java Runtime Environment (JRE) installed on your system. GeoServer requires a **Java 8** environment. The Oracle JRE is preferred, but OpenJDK has been known to work adequately. You can download JRE 8 from Oracle.

   Note: Java 9 is not currently supported.

   Note: For more information about Java and GeoServer, please see the section on *Java Considerations*.

2. Navigate to the GeoServer Download page.

3. Select the version of GeoServer that you wish to download. If you're not sure, select Stable.
4. Click the link for the Windows installer.

Fig. 2.1: Downloading the Windows installer

5. After downloading, double-click the file to launch.
6. At the Welcome screen, click *Next*.

Fig. 2.2: Welcome screen

7. Read the *License* and click *I Agree*.
8. Select the directory of the installation, then click *Next*.
9. Select the Start Menu directory name and location, then click *Next*.
10. Enter the path to a **valid Java Runtime Environment (JRE)**. GeoServer requires a valid JRE in order to run, so this step is required. The installer will inspect your system and attempt to automatically populate this box with a JRE if it is found, but otherwise you will have to enter this path manually. When finished, click *Next*.

Note: A typical path on Windows would be `C:\Program Files\Java\jre8`.

Note: Don't include the `\bin` in the JRE path. So if `java.exe` is located at `C:\Program Files (x86)\Java\jre8\bin\java.exe`, set the path to be `C:\Program Files (x86)\Java\jre8`.

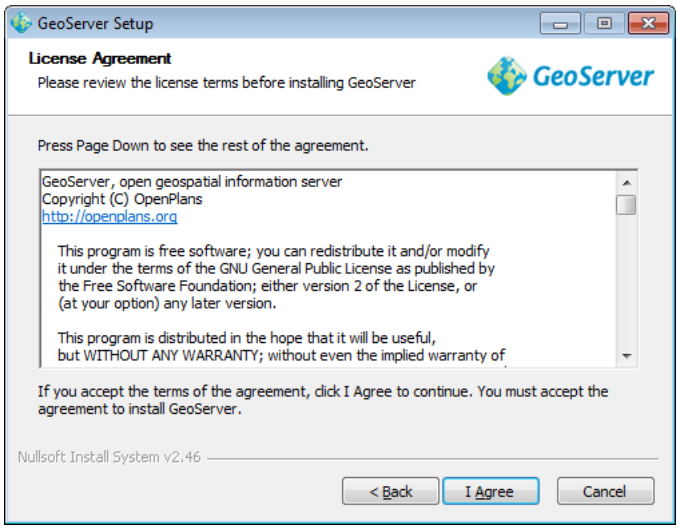


Fig. 2.3: GeoServer license

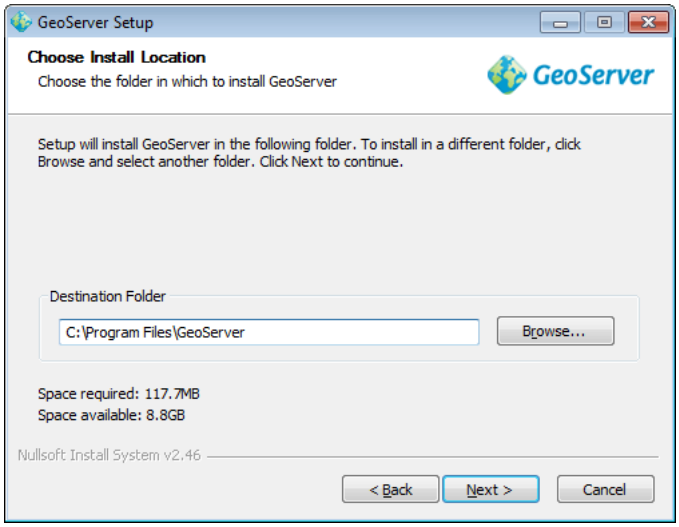


Fig. 2.4: GeoServer install directory

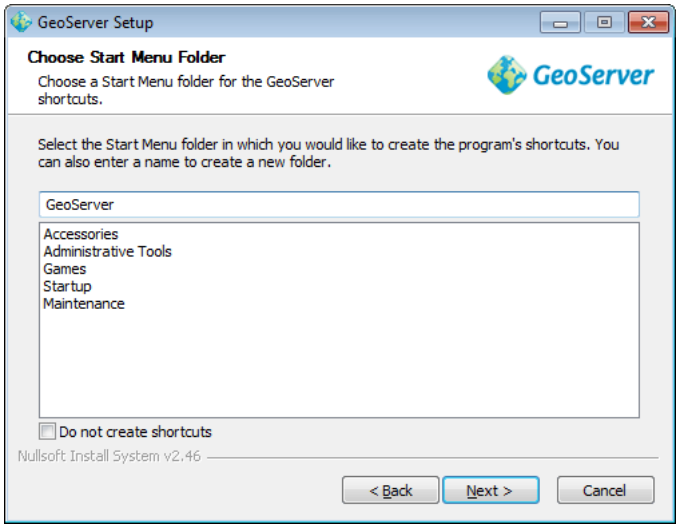


Fig. 2.5: Start menu location

Note: For more information about Java and GeoServer, please see the section on *Java Considerations*.

Fig. 2.6: Selecting a valid JRE

11. Enter the path to your GeoServer data directory or select the default. If this is your first time using GeoServer, select the *Default data directory*. When finished, click *Next*.
12. Enter the username and password for administration of GeoServer. GeoServer's *Web administration interface* requires authentication for management, and what is entered here will become those administrator credentials. The defaults are *admin* / *geoserver*. It is recommended to change these from the defaults. When finished, click *Next*.
13. Enter the port that GeoServer will respond on. This affects the location of the GeoServer *Web administration interface*, as well as the endpoints of the GeoServer services such as *Web Map Service (WMS)* and *Web Feature Service (WFS)*. The default port is *8080*, though any valid and unused port will work. When finished, click *Next*.

Fig. 2.7: Setting a GeoServer data directory

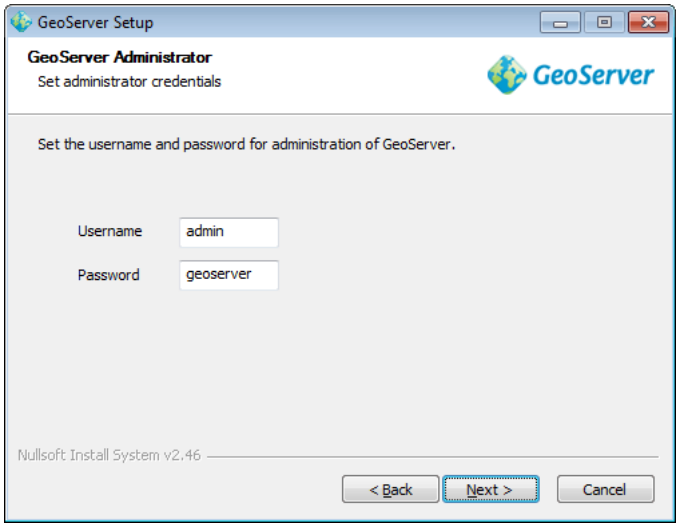


Fig. 2.8: Setting the username and password for GeoServer administration

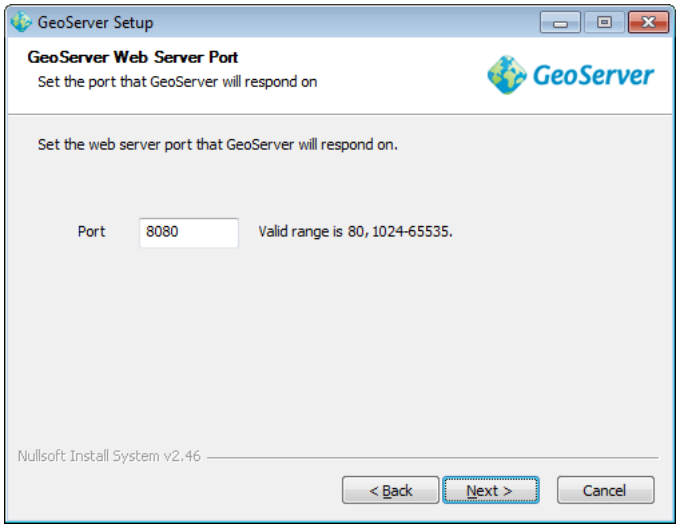


Fig. 2.9: Setting the GeoServer port

14. Select whether GeoServer should be run manually or installed as a service. When run manually, GeoServer is run like a standard application under the current user. When installed as a service, GeoServer is integrated into Windows Services, and thus is easier to administer. If running on a server, or to manage GeoServer as a service, select *Install as a service*. Otherwise, select *Run manually*. When finished, click *Next*.

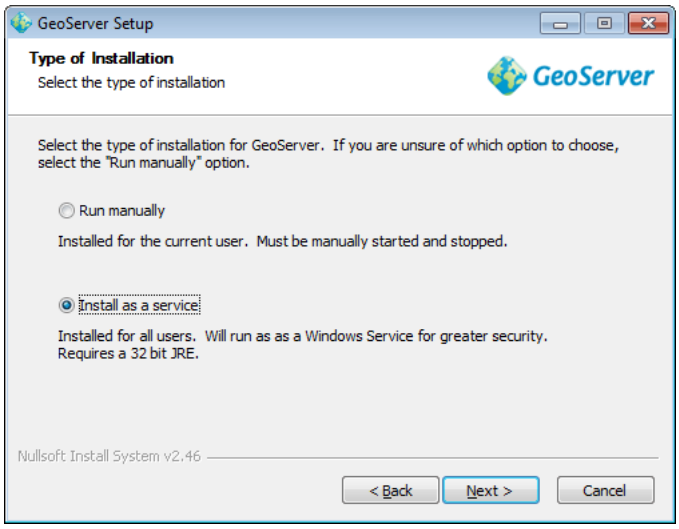


Fig. 2.10: Installing GeoServer as a service

15. Review your selections and click the *Back* button if any changes need to be made. Otherwise, click *Install*.
16. GeoServer will install on your system. When finished, click *Finish* to close the installer.
17. If you installed GeoServer as a service, it is already running. Otherwise, you can start GeoServer by going to the Start Menu, and clicking *Start GeoServer* in the GeoServer folder.
18. Navigate to `http://localhost:8080/geoserver` (or wherever you installed GeoServer) to access the GeoServer *Web administration interface*.

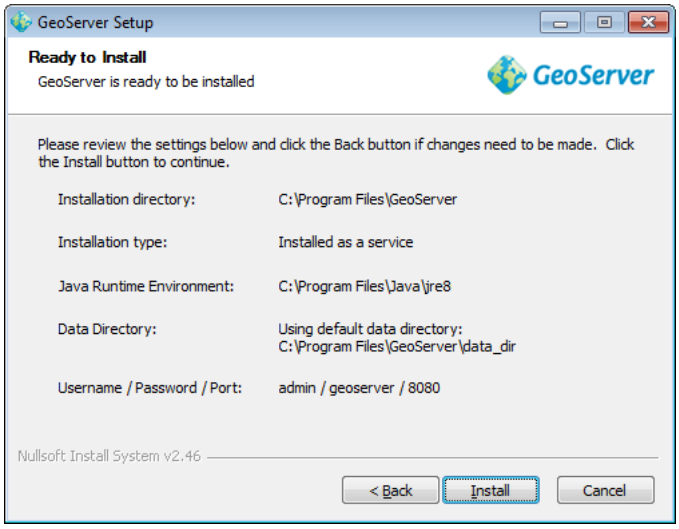


Fig. 2.11: Verifying settings

If you see the GeoServer logo, then GeoServer is successfully installed.

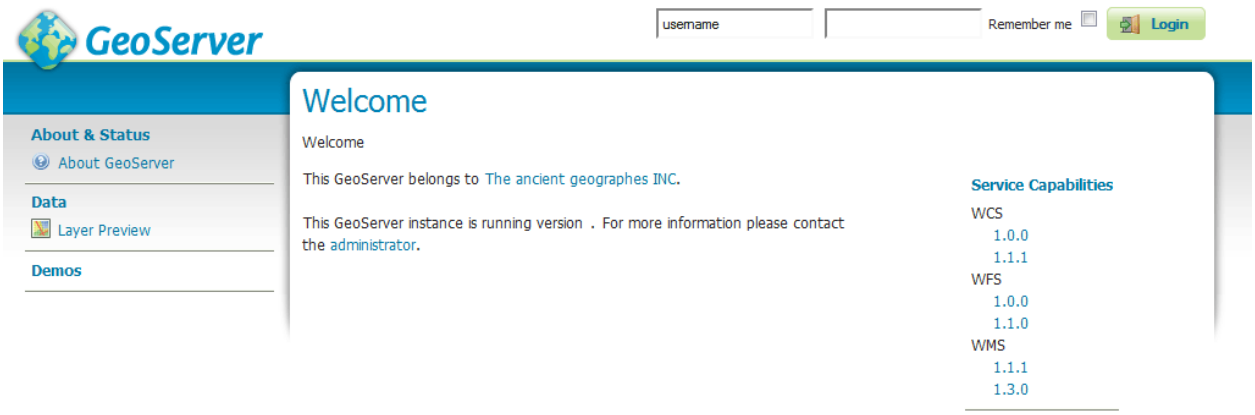


Fig. 2.12: GeoServer installed and running successfully

2.1.1 Uninstallation

GeoServer can be uninstalled in two ways: by running the `uninstall.exe` file in the directory where GeoServer was installed, or by standard Windows program removal.

2.2 Windows binary

Note: For the wizard-based installer on Windows, please see the section on the *Windows installer*. For installing on Windows with an existing application server such as Tomcat, please see the *Web archive* section.

An alternate way of installing GeoServer on Windows is to use the platform-independent binary. This version is a GeoServer web application bundled inside Jetty, a lightweight and portable application server. It has the advantages of working very similarly across all operating systems and is very simple to set up.

2.2.1 Installation

1. Make sure you have a Java Runtime Environment (JRE) installed on your system. GeoServer requires a **Java 8** environment. The Oracle JRE is preferred, but OpenJDK has been known to work adequately. You can download JRE 8 from Oracle.

   Note: Java 9 is not currently supported.

   Note: For more information about Java and GeoServer, please see the section on *Java Considerations*.

2. Navigate to the GeoServer Download page.
3. Select the version of GeoServer that you wish to download. If you're not sure, select Stable.
4. Select *Platform Independent Binary* on the download page.
5. Download the archive and unpack to the directory where you would like the program to be located.

   Note: A suggested location would be `C:\Program Files\GeoServer`.

Setting environment variables

You will need to set the `JAVA_HOME` environment variable if it is not already set. This is the path to your JRE such that `%JAVA_HOME%\bin\java.exe` exists.

1. Navigate to *Control Panel* → *System* → *Advanced* → *Environment Variables*.
2. Under *System variables* click *New*.
3. For *Variable name* enter `JAVA_HOME`. For *Variable value* enter the path to your JDK/JRE.
4. Click OK three times.

Note: You may also want to set the `GEOSERVER_HOME` variable, which is the directory where GeoServer is installed, and the `GEOSERVER_DATA_DIR` variable, which is the location of the GeoServer data directory (which by default is `%GEOSERVER_HOME\data_dir`). The latter is mandatory if you wish to use a data directory other than the default location. The procedure for setting these variables is identical to setting the `JAVA_HOME` variable.

2.2.2 Running

Note: This can be done either via Windows Explorer or the command line.

1. Navigate to the `bin` directory inside the location where GeoServer is installed.

2. Run `startup.bat`. A command-line window will appear and persist. This window contains diagnostic and troubleshooting information. This window must be left open, otherwise GeoServer will shut down.
3. Navigate to `http://localhost:8080/geoserver` (or wherever you installed GeoServer) to access the GeoServer *Web administration interface*.

If you see the GeoServer logo, then GeoServer is successfully installed.

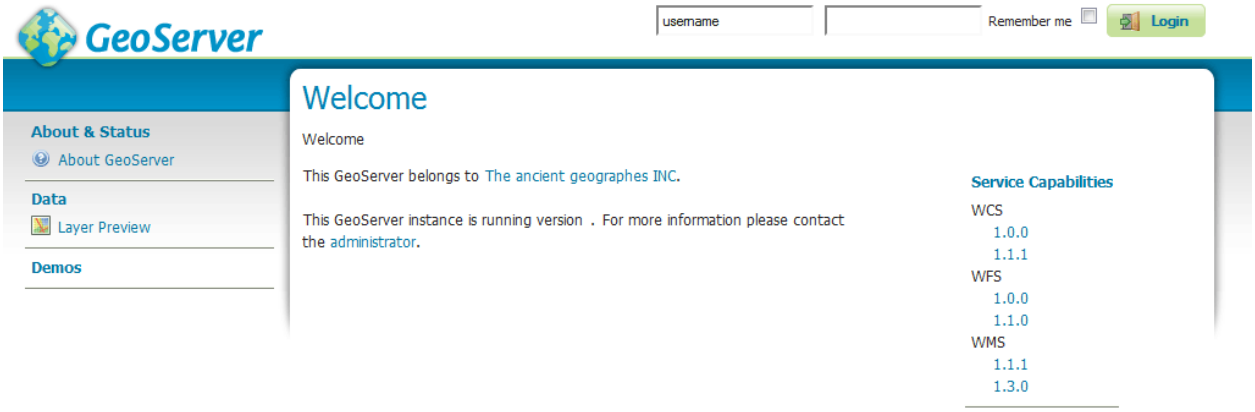


Fig. 2.13: GeoServer installed and running successfully

2.2.3 Stopping

To shut down GeoServer, either close the persistent command-line window, or run the `shutdown.bat` file inside the `bin` directory.

2.2.4 Uninstallation

1. Stop GeoServer (if it is running).
2. Delete the directory where GeoServer is installed.

2.3 Mac OS X installer

The Mac OS X installer provides an easy way to set up GeoServer on your system, as it requires no configuration files to be edited or command line settings.

1. Make sure you have a Java Runtime Environment (JRE) installed on your system. GeoServer requires a **Java 8** environment, and the JRE supplied by OS X is not sufficient. For more information, please see the instructions for installing Oracle Java on OS X.

> **Note:** Java 9 is not currently supported.

Note: For more information about Java and GeoServer, please see the section on *Java Considerations*.

2. Navigate to the GeoServer Download page.
3. Select the version of GeoServer that you wish to download. If you're not sure, select Stable.
4. Click the link for the Mac OS X installer to begin the download.
5. When downloaded, double click on the file to open it.
6. Drag the GeoServer icon to the Applications folder.

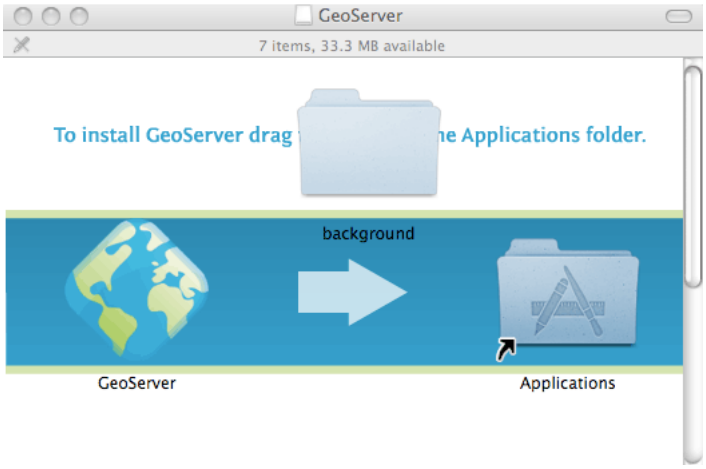


Fig. 2.14: Drag the GeoServer icon to Applications to install

7. Navigate to your Applications folder and double click the GeoServer icon.
8. In the resulting GeoServer console window, start GeoServer by going to *Server* → *Start*.

Fig. 2.15: Starting GeoServer

9. The console window will be populated with log entries showing the GeoServer loading process. Once GeoServer is completely started, a browser window will open at `http://localhost:8080/geoserver`, which is the *Web administration interface* for GeoServer.

Warning: If you encounter the following error during startup, you may have some invalid JAI jars from the default Mac Java install:

```
java.lang.NoClassDefFoundError: Could not initialize class javax.media.
↪jai.JAI
```

To fix this error, locate your Java extensions folder (Usually `/System/Library/Java/Extensions` and/or `~/Library/Java/Extensions`), and delete the following jars:

```
jai_codec-1.1.3.jar
jai_core-1.1.3.jar
jai_imageio-1.1.jar
```

If you have upgraded your OS from an older version, you may not have permission to delete these jars. In this case, you will first need to disable System Integrity Protection.

2.4 Mac OS X binary

Note: For the installer on OS X, please see the section on the *Mac OS X installer*. For installing on OS X with an existing application server such as Tomcat, please see the *Web archive* section.

An alternate way of installing GeoServer on OS X is to use the platform-independent binary. This version is a GeoServer web application bundled inside Jetty, a lightweight and portable application server. It has the advantages of working very similarly across all operating systems and is very simple to set up.

2.4.1 Installation

1. Make sure you have a Java Runtime Environment (JRE) installed on your system. GeoServer requires a **Java 8** environment, and the JRE supplied by OS X is not sufficient. For more information, please see the instructions for installing Oracle Java on OS X.

   Note: Java 9 is not currently supported.

   Note: For more information about Java and GeoServer, please see the section on *Java Considerations*.

2. Navigate to the GeoServer Download page.
3. Select the version of GeoServer that you wish to download. If you're not sure, select Stable.
4. Select *Platform Independent Binary* on the download page.
5. Download the archive and unpack to the directory where you would like the program to be located.

   Note: A suggested location would be `/usr/local/geoserver`.

6. Add an environment variable to save the location of GeoServer by typing the following command:

   ```
   echo "export GEOSERVER_HOME=/usr/local/geoserver" >> ~/.profile
   . ~/.profile
   ```

7. Make yourself the owner of the `geoserver` folder, by typing the following command:

   ```
   sudo chown -R <USERNAME> /usr/local/geoserver/
   ```

where `USER_NAME` is your user name

8. Start GeoServer by changing into the directory `geoserver/bin` and executing the `startup.sh` script:

```
cd geoserver/bin
sh startup.sh
```

> **Warning:** If you encounter the following error during startup, you may have some invalid JAI jars from the default Mac Java install:
>
> ```
> java.lang.NoClassDefFoundError: Could not initialize class javax.media.
> ↪jai.JAI
> ```
>
> To fix this error, locate your Java extensions folder (Usually `/System/Library/Java/Extensions` and/or `~/Library/Java/Extensions`), and delete the following jars:
>
> ```
> jai_codec-1.1.3.jar
> jai_core-1.1.3.jar
> jai_imageio-1.1.jar
> ```
>
> If you have upgraded your OS from an older version, you may not have permission to delete these jars. In this case, you will first need to disable System Integrity Protection.

9. In a web browser, navigate to `http://localhost:8080/geoserver`.

If you see the GeoServer logo, then GeoServer is successfully installed.

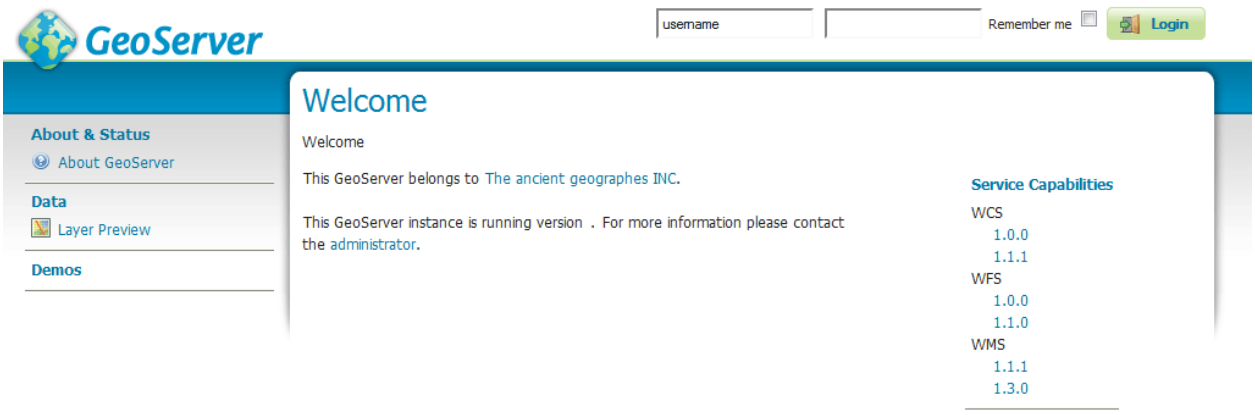


Fig. 2.16: GeoServer installed and running successfully

To shut down GeoServer, either close the persistent command-line window, or run the `shutdown.sh` file inside the `bin` directory.

2.4.2 Uninstallation

1. Stop GeoServer (if it is running).
2. Delete the directory where GeoServer is installed.

2.5 Linux binary

Note: For installing on Linux with an existing application server such as Tomcat, please see the *Web archive* section.

The platform-independent binary is a GeoServer web application bundled inside Jetty, a lightweight and portable application server. It has the advantages of working very similarly across all operating systems and is very simple to set up.

2.5.1 Installation

1. Make sure you have a Java Runtime Environment (JRE) installed on your system. GeoServer requires a **Java 8** environment. The Oracle JRE is preferred, but OpenJDK has been known to work adequately. You can download JRE 8 from Oracle.

   Note: Java 9 is not currently supported.

2. Select the version of GeoServer that you wish to download. If you're not sure, select Stable.
3. Select *Platform Independent Binary* on the download page.
4. Download the archive and unpack to the directory where you would like the program to be located.

   Note: A suggested location would be `/usr/share/geoserver`.

5. Add an environment variable to save the location of GeoServer by typing the following command:

   ```
   echo "export GEOSERVER_HOME=/usr/share/geoserver" >> ~/.profile
   . ~/.profile
   ```

6. Make yourself the owner of the `geoserver` folder. Type the following command in the terminal window, replacing `USER_NAME` with your own username :

   ```
   sudo chown -R USER_NAME /usr/share/geoserver/
   ```

7. Start GeoServer by changing into the directory `geoserver/bin` and executing the `startup.sh` script:

   ```
   cd geoserver/bin
   sh startup.sh
   ```

8. In a web browser, navigate to `http://localhost:8080/geoserver`.

If you see the GeoServer logo, then GeoServer is successfully installed.

To shut down GeoServer, either close the persistent command-line window, or run the `shutdown.sh` file inside the `bin` directory.

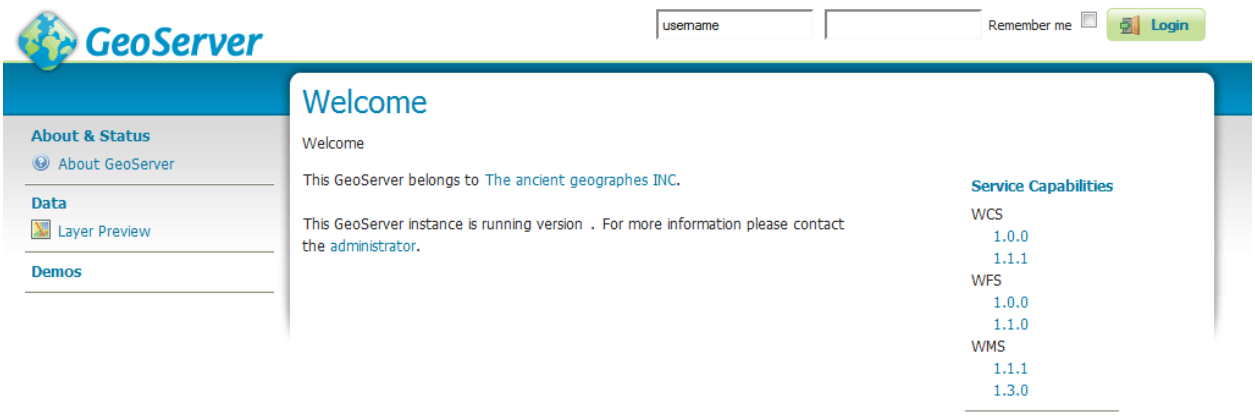


Fig. 2.17: GeoServer installed and running successfully

2.5.2 Uninstallation

1. Stop GeoServer (if it is running).
2. Delete the directory where GeoServer is installed.

2.6 Web archive

GeoServer is packaged as a standalone servlet for use with existing application servers such as Apache Tomcat and Jetty.

Note: GeoServer has been mostly tested using Tomcat, and so is the recommended application server. GeoServer requires a newer version of Tomcat (7.0.65 or later) that implements Servlet 3 and annotation processing. Other application servers have been known to work, but are not guaranteed.

2.6.1 Installation

1. Make sure you have a Java Runtime Environment (JRE) installed on your system. GeoServer requires a **Java 8** environment. The Oracle JRE is preferred, but OpenJDK has been known to work adequately. You can download JRE 8 from Oracle.

   Note: Java 9 is not currently supported.

   Note: For more information about Java and GeoServer, please see the section on *Java Considerations*.

   ---

2. Navigate to the GeoServer Download page.
3. Select *Web Archive* on the download page.
4. Download and unpack the archive.

5. Deploy the web archive as you would normally. Often, all that is necessary is to copy the `geoserver.war` file to the application server's `webapps` directory, and the application will be deployed.

   Note: A restart of your application server may be necessary.

2.6.2 Running

Use your container application's method of starting and stopping webapps to run GeoServer.

To access the *Web administration interface*, open a browser and navigate to `http://SERVER/geoserver`. For example, with Tomcat running on port 8080 on localhost, the URL would be `http://localhost:8080/geoserver`.

2.6.3 Uninstallation

1. Stop the container application.
2. Remove the GeoServer webapp from the container application's `webapps` directory. This will usually include the `geoserver.war` file as well as a `geoserver` directory.

2.7 Upgrading existing versions

Warning: Be aware that some upgrades are not reversible, meaning that the data directory may be changed so that it is no longer compatible with older versions of GeoServer. See *Migrating a data directory between versions* for more details.

The general GeoServer upgrade process is as follows:

1. Back up the current data directory. This can involve simply copying the directory to an additional place.
2. Make sure that the current data directory is external to the application (not located inside the application file structure).
3. Uninstall the old version and install the new version.

   Note: Alternately, you can install the new version directly over top of the old version.

4. Make sure that the new version continues to point to the same data directory used by the previous version.

2.7.1 Notes on upgrading specific versions

GeoJSON encoding (GeoServer 2.6 and newer)

As of GeoServer 2.6, the GeoJSON produced by the WFS service no longer uses a non-standard encoding for the CRS. To reenable this behavior for compatibility purposes, set

`GEOSERVER_GEOJSON_LEGACY_CRS=true` as a system property, context parameter, or environment variable.

JTS Type Bindings (GeoServer 2.14 and newer)

As of GeoServer 2.14, the output produced by *REST* featuretype and structured coverage requests using a different package name (`org.locationtech` instead of `com.vividsolutions`) for geometry type bindings, due to the upgrade to JTS (Java Topology Suite) 1.16.0. For example:

Before:

```
...
<attribute>
  <name>geom</name>
  <minOccurs>0</minOccurs>
  <maxOccurs>1</maxOccurs>
  <nillable>true</nillable>
  <binding>com.vividsolutions.jts.geom.Point</binding>
</attribute>
...
```

After:

```
...
<attribute>
  <name>geom</name>
  <minOccurs>0</minOccurs>
  <maxOccurs>1</maxOccurs>
  <nillable>true</nillable>
  <binding>org.locationtech.jts.geom.Point</binding>
</attribute>
...
```

Any REST clients which rely on this binding information should be updated to support the new names.

CHAPTER 3

Getting started

This section contains short tutorials for common tasks in GeoServer to get new users using the system quickly.

3.1 Using the web administration interface

GeoServer has a browser-based web administration interface application used to configure all aspects of GeoServer, from adding and publishing data to changing service settings.

The web admin interface is accessed via a web browser at:

```
http://<host>:<port>/geoserver
```

For a default installation on a server the link is:

```
http://localhost:8080/geoserver
```

When the application starts, it displays the Welcome page.

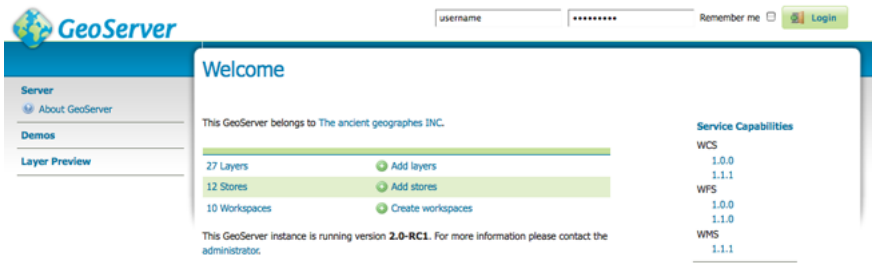


Fig. 3.1: Welcome Page

3.1.1 Logging In

In order to change any server settings or configure data, a user must first be authenticated.

1. Navigate to the upper right of the web interface to log into GeoServer. The default administration credentials are:
   - User name: `admin`
   - Password: `geoserver`

Note: These can be changed in the *Security* section.

Fig. 3.2: Login

2. Once logged in, the Welcome screen changes to show the available admin functions. These are primarily shown in the menus on the left side of the page.

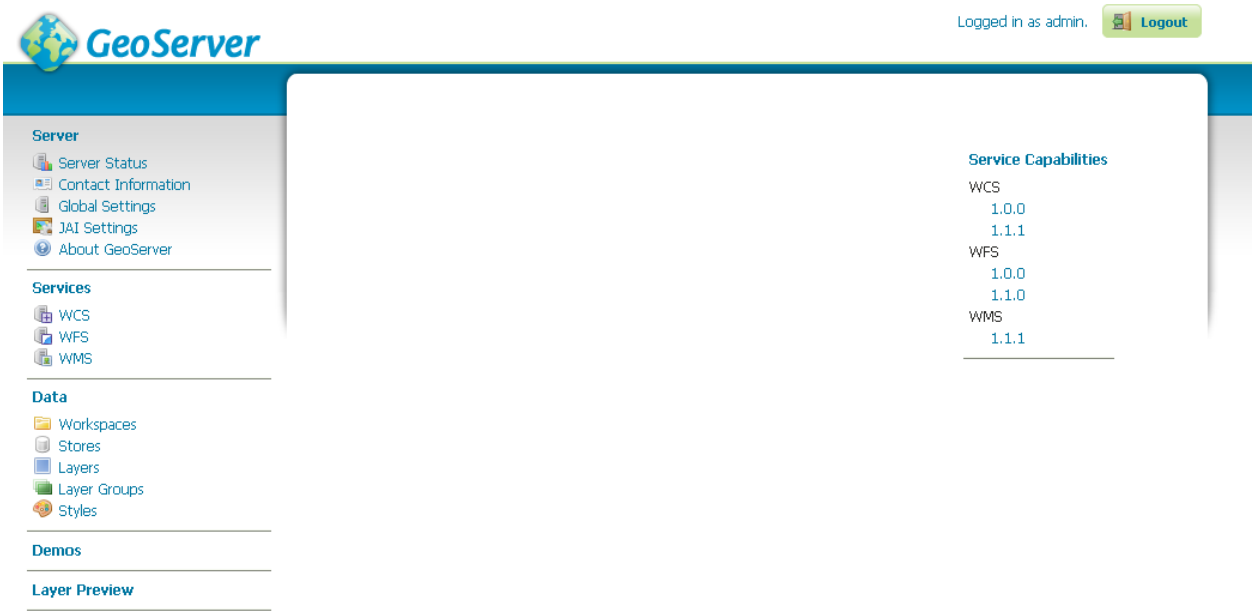
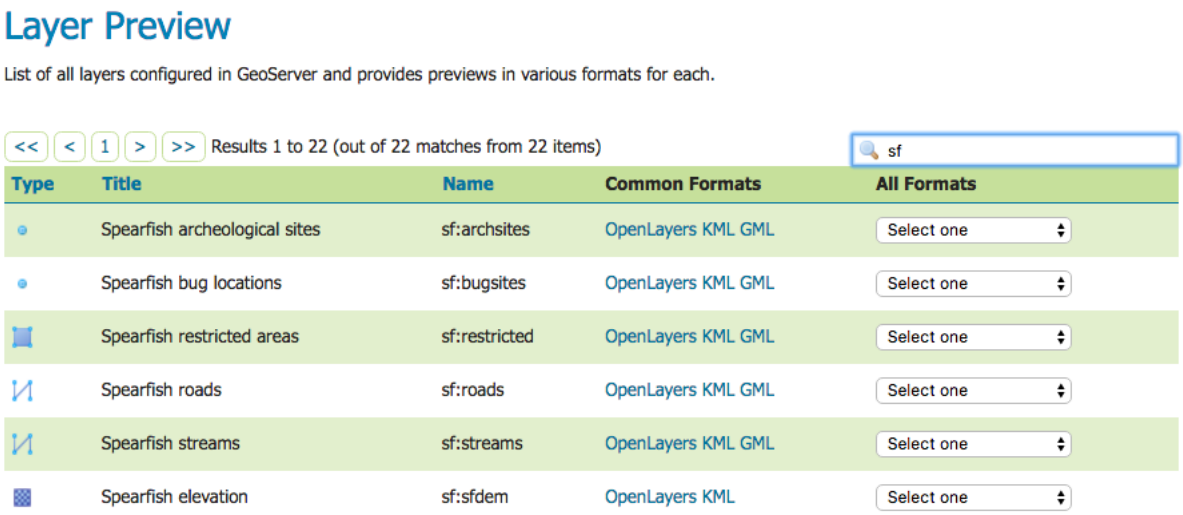


Fig. 3.3: Additional options when logged in

3.1.2 Layer Preview

The *Layer Preview* page allows you to quickly view the output of published layers.

1. Click the *Layer Preview* link on the menu to go to this page.
2. From here, you can find the layer you'd like to preview and click a link for an output format. Click the *OpenLayers* link for a given layer and the view will display.
3. To sort a column alphabetically, click the column header.



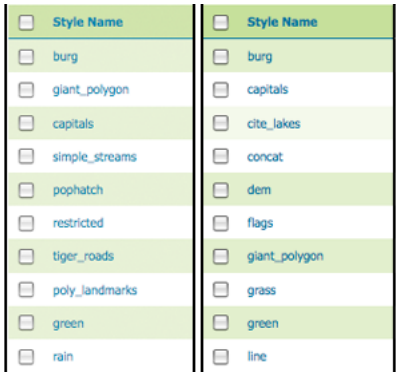


Fig. 3.4: Unsorted (left) and sorted (right) columns

4. Searching can be used to filter the number of items displayed. This is useful for working with data types that contain a large number of items. To search data type items, enter the search string in the search box and click Enter. GeoServer will search the data type for items that match your query, and display a list view showing the search results.

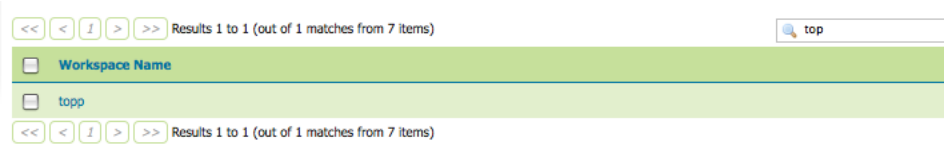


Fig. 3.5: Search results for the query "top" on the Workspace page

Note: Sorting and searching apply to all data configuration pages.

Note: For more information, please see the *Layer Preview* section.

3.2 Publishing a shapefile

This tutorial walks through the steps of publishing a Shapefile with GeoServer.

Note: This tutorial assumes that GeoServer is running at `http://localhost:8080/geoserver`.

3.2.1 Data preparation

First let's gather that the data that we'll be publishing.

1. Download the file `nyc_roads.zip`. This archive contains a shapefile of roads from New York City that will be used during in this tutorial.
2. Unzip the `nyc_roads.zip` into a new directory named `nyc_roads`. The archive contains the following four files:

```
nyc_roads.shp
nyc_roads.shx
nyc_roads.dbf
nyc_roads.prj
```

3. Move the `nyc_roads` directory into `<GEOSERVER_DATA_DIR>/data`, where `<GEOSERVER_DATA_DIR>` is the root of the *GeoServer data directory*. If no changes have been made to the GeoServer file structure, the path is `geoserver/data_dir/data/nyc_roads`.

3.2.2 Creating a new workspace

The next step is to create a workspace for the shapefile. A workspace is a container used to group similar layers together.

Note: This step is optional if you'd like to use an existing workspace. Usually, a workspace is created for each project, which can include stores and layers that are related to each other.

1. In a web browser, navigate to `http://localhost:8080/geoserver`.
2. Log into GeoServer as described in the *Logging In* section.
3. Navigate to *Data* → *Workspaces*.

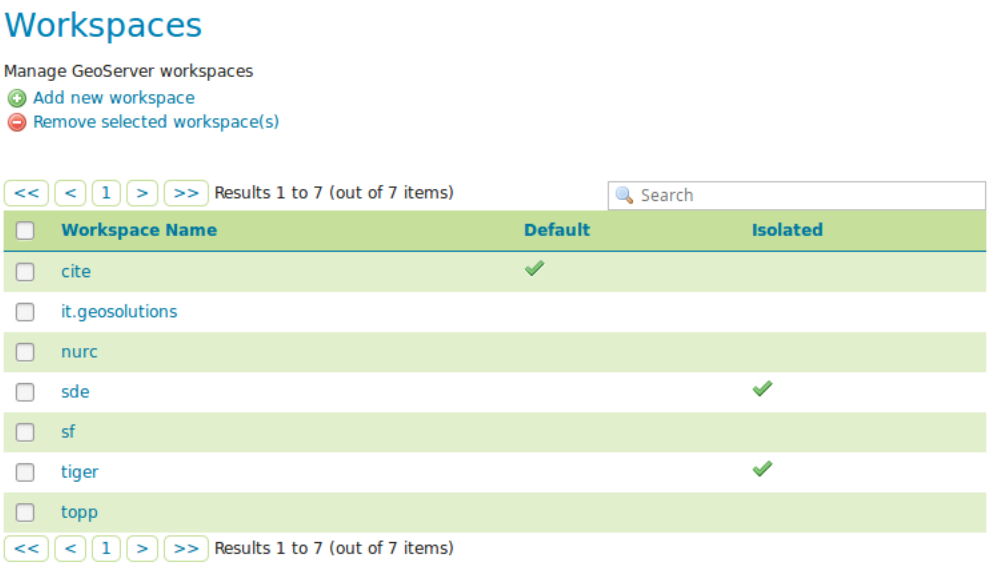


Fig. 3.6: Workspaces page

4. Click the *Add new workspace* button.
5. You will be prompted to enter a workspace *Name* and *Namespace URI*.

Fig. 3.7: Configure a new workspace

6. Enter the *Name* as `nyc` and the *Namespace URI* as `http://geoserver.org/nyc`.

Note: A workspace name is a identifier describing your project. It must not exceed ten characters or contain spaces. A Namespace URI (Uniform Resource Identifier) can usually be a URL associated with your project with an added trailing identifier indicating the workspace. The Namespace URI filed does not need to resolve to an actual valid web address.

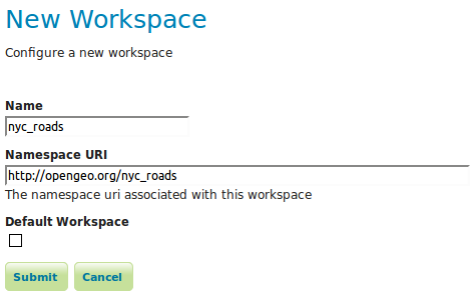


Fig. 3.8: nyc workspace

7. Click the *Submit* button. The `nyc` workspace will be added to the *Workspaces* list.

3.2.3 Create a store

Once the workspace is created, we are ready to add a new store. The store tells GeoServer how to connect to the shapefile.

1. Navigate to *Data→Stores*.
2. You should see a list of stores, including the type of store and the workspace that the store belongs to.
3. In order to add the shapefile, you need to create a new store. Click the *Add new Store* button. You will be redirected to a list of the data sources supported by GeoServer. Note that the data sources are extensible, so your list may look slightly different.

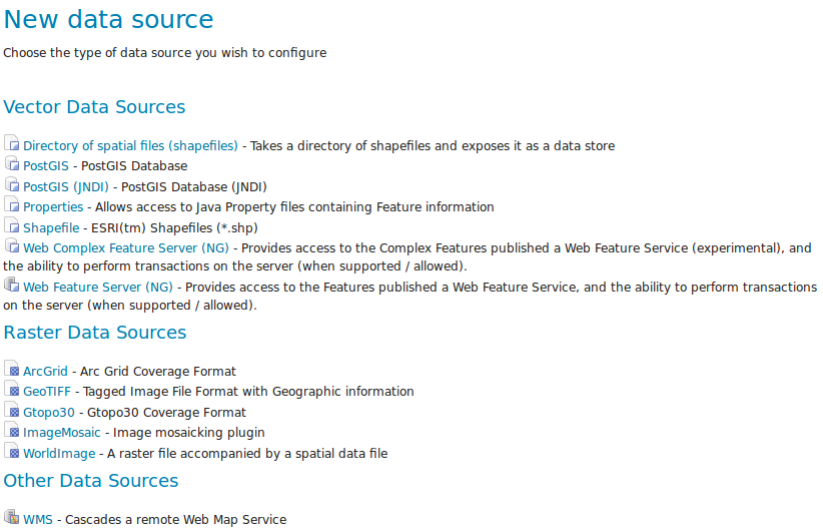


Fig. 3.9: Stores

4. Click *Shapefile*. The *New Vector Data Source* page will display.
5. Begin by configuring the *Basic Store Info*.
   - Select the workspace `nyc` from the drop down menu.
   - Enter the *Data Source Name* as `NYC Roads`
   - Enter a brief *Description* (such as "Roads in New York City").

6. Under *Connection Parameters*, browse to the location *URL* of the shapefile, typically `nyc_roads/nyc_roads.shp`.

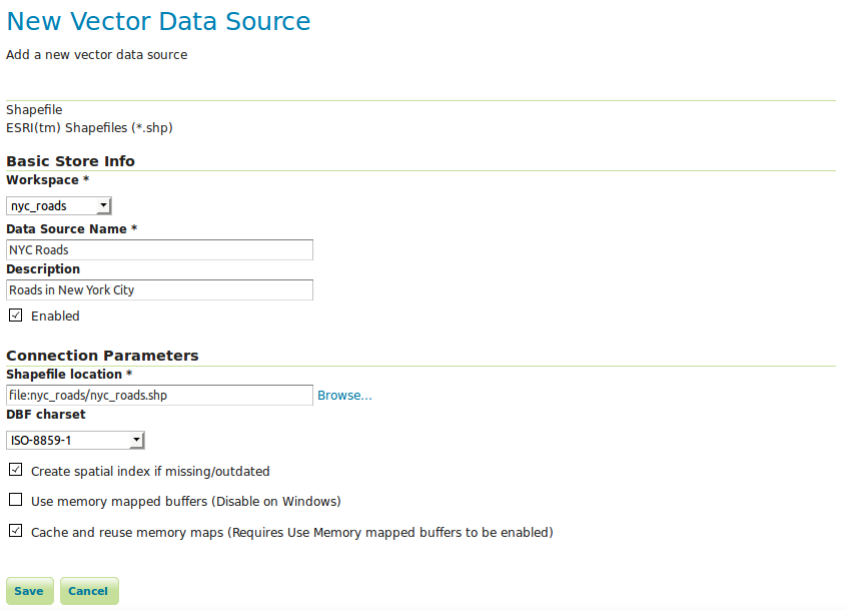


Fig. 3.10: Basic Store Info and Connection Parameters

7. Click *Save*. You will be redirected to the *New Layer* page in order to configure the `nyc_roads` layer.

3.2.4 Creating a layer

Now that the store is loaded, we can publish the layer.

1. On the *New Layer* page, click *Publish* beside the `nyc_roads` layer name.

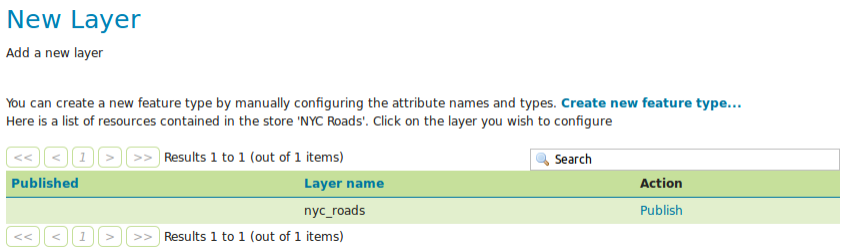


Fig. 3.11: New layer

2. The *Edit Layer* page defines the data and publishing parameters for a layer. Enter a short *Title* and an *Abstract* for the `nyc_roads` layer.
3. Generate the layer's bounding boxes by clicking the *Compute from data* and then *Compute from native bounds* links.
4. Click the *Publishing* tab at the top of the page.
5. We can set the layer's style here. Under *WMS Settings*, ensure that the *Default Style* is set to *line*.
6. Finalize the layer configuration by scrolling to the bottom of the page and clicking *Save*.

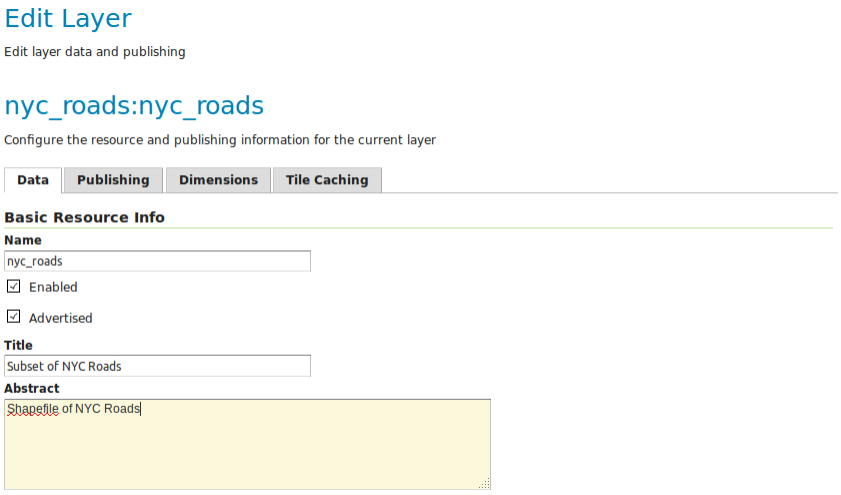


Fig. 3.12: Basic Resource Information

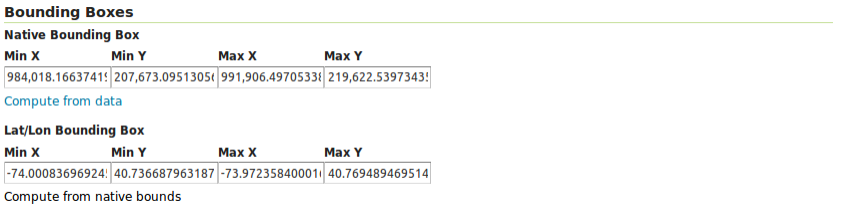


Fig. 3.13: Generating bounding boxes

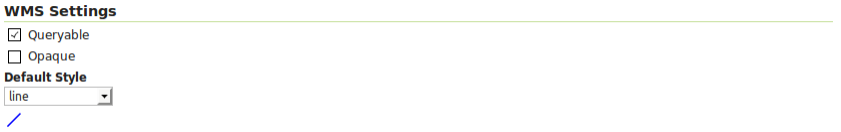


Fig. 3.14: Select Default Style

3.2.5 Previewing the layer

In order to verify that the `nyc_roads` layer is published correctly, we can preview the layer.

1. Navigate to the *Layer Preview* screen and find the `nyc:nyc_roads` layer.

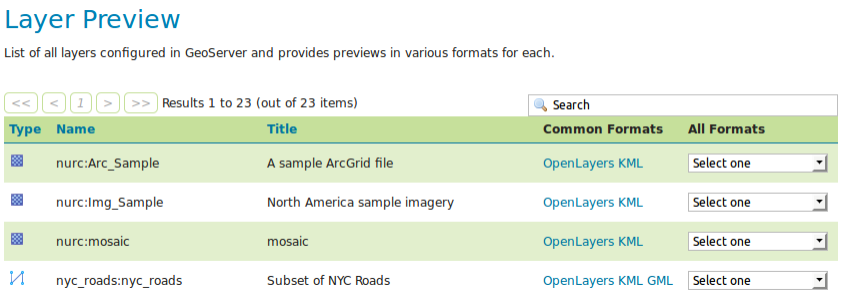


Fig. 3.15: Layer Preview

2. Click the *OpenLayers* link in the *Common Formats* column.
3. An OpenLayers map will load in a new tab and display the shapefile data with the default line style. You can use this preview map to zoom and pan around the dataset, as well as display the attributes of features.

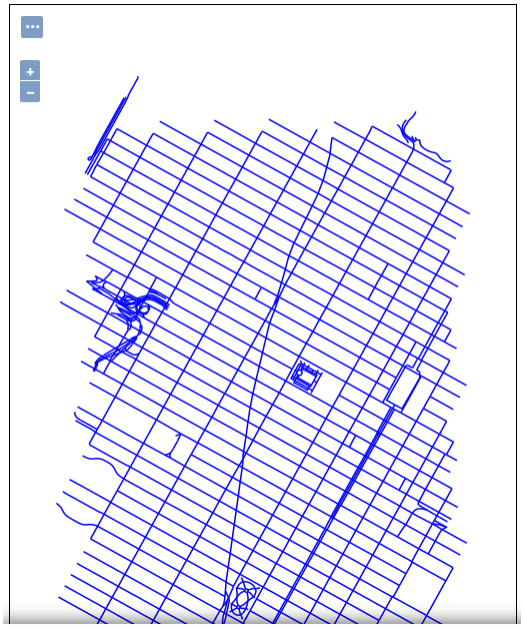


Fig. 3.16: Preview map of nyc_roads

3.3 Publishing a PostGIS table

This tutorial walks through the steps of publishing a PostGIS table with GeoServer.

Note: This tutorial assumes that PostgreSQL/PostGIS has been previously installed on the system and responding on `localhost` on port `5432`, and also that GeoServer is running at `http://localhost:8080/geoserver`.

3.3.1 Data preparation

First let's gather that the data that we'll be publishing.

1. Download the file `nyc_buildings.zip`. It contains a PostGIS dump of a dataset of buildings from New York City.
2. Create a PostGIS database called `nyc`. This can be done with the following commands:

```
createdb nyc
psql -d nyc -c 'CREATE EXTENSION postgis'
```

   Note: You may need to supply a user name and password with these commands.

3. Extract `nyc_buildings.sql` from `nyc_buildings.zip`.
4. Import `nyc_buildings.sql` into the `nyc` database:

```
psql -f nyc_buildings.sql nyc
```

3.3.2 Creating a new workspace

The next step is to create a workspace for the data. A workspace is a container used to group similar layers together.

Note: This step is optional if you'd like to use an existing workspace. Usually, a workspace is created for each project, which can include stores and layers that are related to each other.

1. In a web browser, navigate to `http://localhost:8080/geoserver`.
2. Log into GeoServer as described in the *Logging In* section.
3. Navigate to *Data* → *Workspaces*.
4. Click the *Add new workspace* button.
5. You will be prompted to enter a workspace *Name* and *Namespace URI*.
6. Enter the *Name* as `nyc` and the *Namespace URI* as `http://geoserver.org/nyc`.

   Note: A workspace name is a identifier describing your project. It must not exceed ten characters or contain spaces. A Namespace URI (Uniform Resource Identifier) can usually be a URL associated with your project with an added trailing identifier indicating the workspace. The Namespace URI filed does not need to resolve to an actual valid web address.

7. Click the *Submit* button. The `nyc` workspace will be added to the *Workspaces* list.

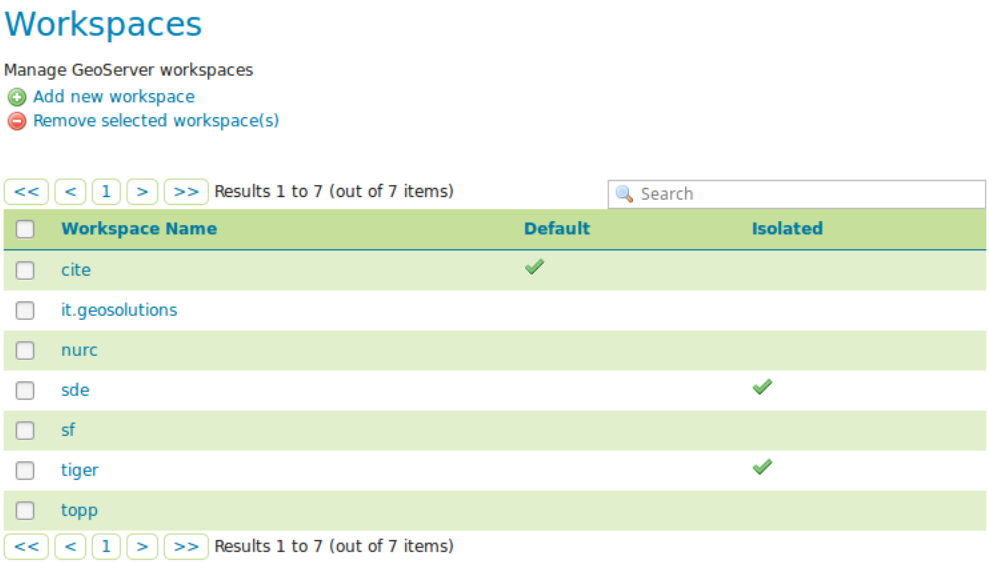


Fig. 3.17: Workspaces page

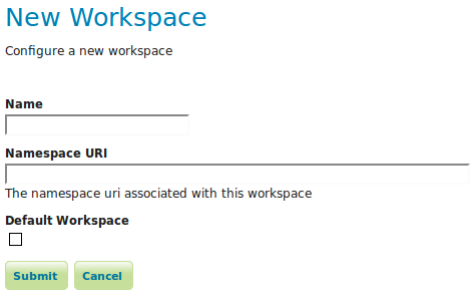


Fig. 3.18: Configure a new workspace

3.3.3 Creating a store

Once the workspace is created, we are ready to add a new store. The store tells GeoServer how to connect to the shapefile.

1. Navigate to *Data→Stores*.
2. You should see a list of stores, including the type of store and the workspace that the store belongs to.

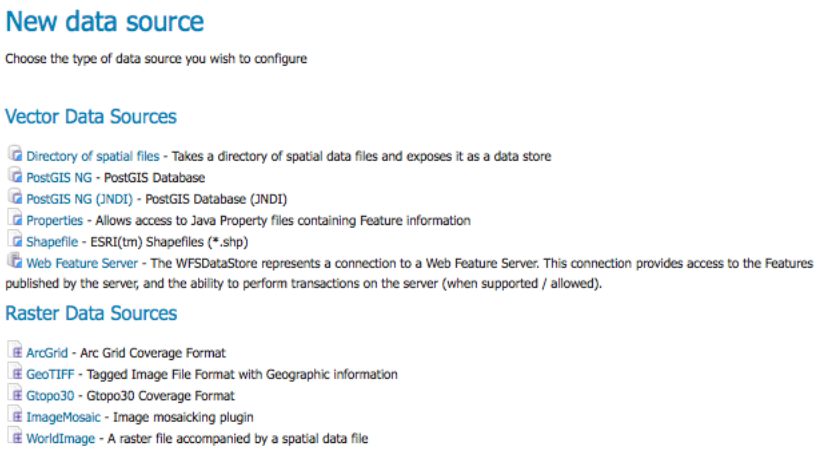


Fig. 3.19: Adding a new data source

3. Create a new store by clicking the `PostGIS` link.
4. Enter the *Basic Store Info*:
   - Select the `nyc` *Workspace*
   - Enter the *Data Source Name* as `nyc_buildings`
   - Add a brief *Description*

Fig. 3.20: Basic Store Info

5. Specify the PostGIS database *Connection Parameters*:

Option	Value
dbtype	`postgis`
host	`localhost`
port	`5432`
database	`nyc`
schema	`public`
user	`postgres`
passwd	(Password for the `postgres` user)
validate connections	(Checked)

Note: Leave all other fields at their default values.

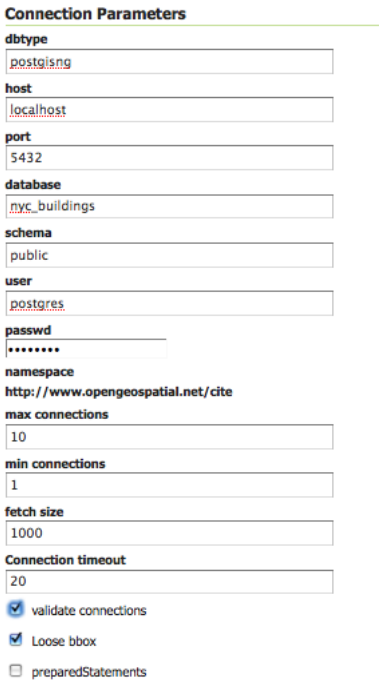


Fig. 3.21: Connection Parameters

6. Click *Save*.

3.3.4 Creating a layer

Now that the store is loaded, we can publish the layer.

1. Navigate to *Data* → *Layers*.
2. Click *Add a new resource*.
3. From the *New Layer chooser* menu, select `nyc:nyc_buidings`.
4. On the resulting layer row, select the layer name `nyc_buildings`.
5. The *Edit Layer* page defines the data and publishing parameters for a layer. Enter a short *Title* and an *Abstract* for the `nyc_buildings` layer.

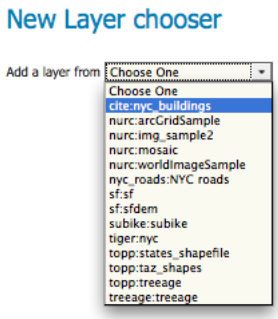


Fig. 3.22: Store selection

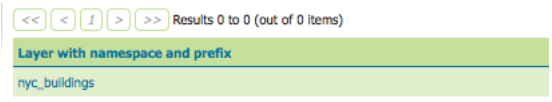


Fig. 3.23: New layer selection

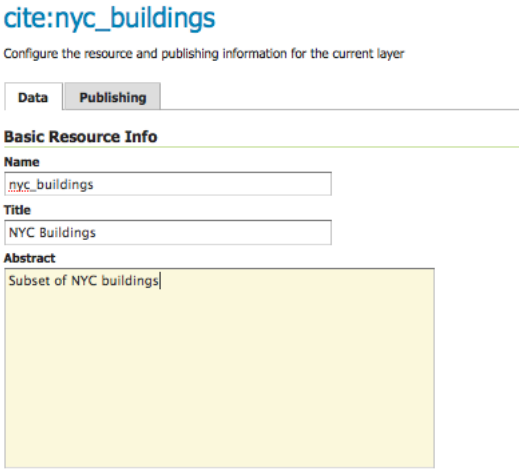


Fig. 3.24: Basic Resource Info

6. Generate the layer's bounding boxes by clicking the *Compute from data* and then *Compute from native bounds* links.

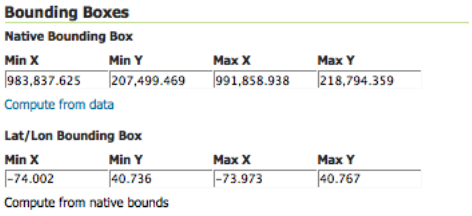


Fig. 3.25: Generating bounding boxes

7. Click the *Publishing* tab at the top of the page.
8. We can set the layer's style here. Under *WMS Settings*, ensure that the *Default Style* is set to *polygon*.

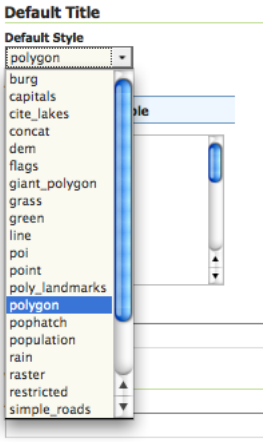


Fig. 3.26: Select Default Style

9. Finalize the layer configuration by scrolling to the bottom of the page and clicking *Save*.

3.3.5 Previewing the layer

In order to verify that the `nyc_buildings` layer is published correctly, we can preview the layer.

1. Navigate to the *Layer Preview* screen and find the `nyc:nyc_buildings` layer.
2. Click the *OpenLayers* link in the *Common Formats* column.
3. An OpenLayers map will load in a new tab and display the shapefile data with the default line style. You can use this preview map to zoom and pan around the dataset, as well as display the attributes of features.

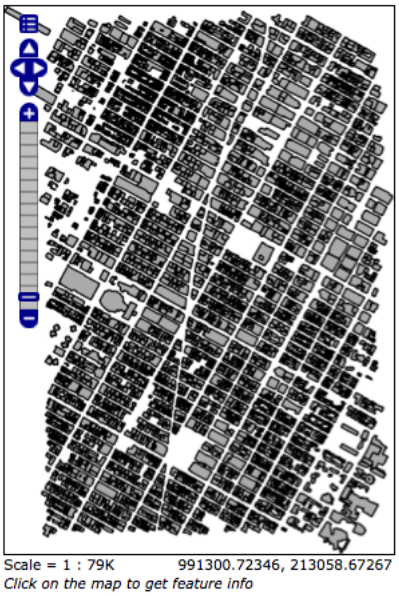


Fig. 3.27: Preview map of nyc_buildings

CHAPTER 4

Web administration interface

The Web administration interface is a web-based tool for configuring all aspects of GeoServer, from adding data to changing service settings. In a default GeoServer installation, this interface is accessed via a web browser at `http://localhost:8080/geoserver/web`. However, this URL may vary depending on your local installation.

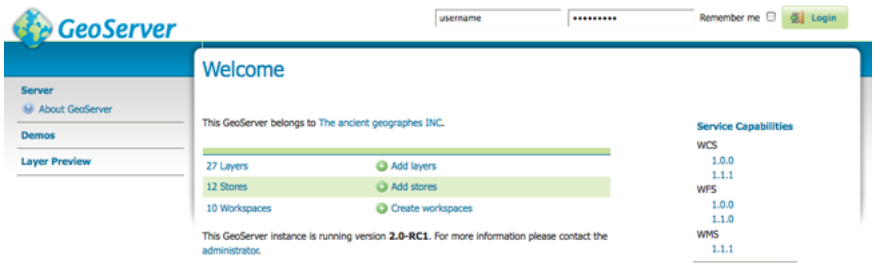


Fig. 4.1: Web admin interface

The following sections detail the menu options available in GeoServer. **Unless otherwise specified, you will need to be logged in with administrative credentials to see these options.**

4.1 About & Status

The *About & Status* section provides access to GeoServer diagnostic and configuration tools, and can be particularly useful for debugging.

- The *Status* page shows a summary of server configuration parameters and run-time status.
- The *GeoServer Logs* page shows the GeoServer log output. This is useful for determining errors without having to leave the browser.

- The *Contact Information* page sets the public contact information available in the Capabilities document of the WMS server.
- The *About GeoServer* section provides links to the GeoServer documentation, homepage and bug tracker. **You do not need to be logged into GeoServer to access this page.**

4.2 Data

The *Data management* section contains configuration options for all the different data-related settings.

- The *Layer Preview* page provides links to layer previews in various output formats, including the common OpenLayers and KML formats. This page helps to visually verify and explore the configuration of a particular layer. **You do not need to be logged into GeoServer to access the Layer Preview.**
- The *Workspaces* page displays a list of workspaces, with the ability to add, edit, and delete. Also shows which workspace is the default for the server.
- The *Stores* page displays a list of stores, with the ability to add, edit, and delete. Details include the workspace associated with the store, the type of store (data format), and whether the store is enabled.
- The *Layers* page displays a list of layers, with the ability to add, edit, and delete. Details include the workspace and store associated with the layer, whether the layer is enabled, and the spatial reference system (SRS) of the layer.
- The *Layer Groups* page displays a list of layer groups, with the ability to add, edit, and delete. Details include the associated workspace (if any).
- The *Styles* page displays a list of styles, with the ability to add, edit, and delete. Details include the associated workspace (if any).

In each of these pages that contain a table, there are three different ways to locate an object: sorting, searching, and paging. To alphabetically sort a data type, click on the column header. For simple searching, enter the search criteria in the search box and hit Enter. And to page through the entries (25 at a time), use the arrow buttons located on the bottom and top of the table.

4.3 Services

The *Services* section is for configuring the services published by GeoServer.

- The *Web Coverage Service (WCS)* page manages metadata, resource limits, and SRS availability for WCS.
- The *Web Feature Service (WFS)* page manages metadata, feature publishing, service level options, and data-specific output for WFS.
- The *Web Map Service (WMS)* page manages metadata, resource limits, SRS availability, and other data-specific output for WMS.

4.4 Settings

The *Settings* section contains configuration settings that apply to the entire server.

- The *Global Settings* page configures messaging, logging, character and proxy settings for the entire server.

- The *Image Processing* page configures several JAI parameters, used by both WMS and WCS operations.
- The *Coverage Access* page configures settings related to loading and publishing coverages.

4.5 Tile Caching

The *Tile Caching* section configures the embedded *GeoWebCache*.

- The *Tile Layers* page shows which layers in GeoServer are also available as tiled (cached)layers, with the ability to add, edit, and delete.
- The *Caching Defaults* page sets the global options for the caching service.
- The *Gridsets* page shows all available gridsets for the tile caches, with the ability to add, edit, and delete.
- The *Disk Quota* page sets the options for tile cache management on disk, including strategies to reduce file size when necessary.
- The *BlobStores* pages manages the different blobstores (tile cache sources) known to the embedded GeoWebCache.

4.6 Security

The *Security* section configures the built-in *security subsystem*.

- The *Settings* page manages high-level options for the security subsystem.
- The *Authentication* page manages authentication filters, filter chains, and providers.
- The *Passwords* page manages the password policies for users and the master (root) account.
- The *Users, Groups, Roles* page manages the users, groups, and roles, and how they are all associated with each other. Passwords for user accounts can be changed here.
- The *Data* page manages the data-level security options, allowing workspaces and layers to be restricted by role.
- The *Services* page manages the service-level security options, allowing services and operations to be restricted by role.

4.7 Demos

The *Demos* section contains links to example WMS, WCS, and WFS requests for GeoServer as well as a listing all SRS info known to GeoServer. In addition, there is a reprojection console for converting coordinates between spatial reference systems, and a request builder for WCS requests. **You do not need to be logged into GeoServer to access these pages.**

4.8 Tools

The *Tools* section contains administrative tools. By default, the only tool is the *Catalog Bulk Load Tool*, which can bulk copy test data into the catalog.

4.9 Extensions

GeoServer extensions can add functionality and extra options to the web interface. Details can be found in the section for each extension.

CHAPTER 5

Data management

GeoServer connects to and publishes data from a wide variety of sources. This section will discuss how to use the GeoServer web interface to accomplish most common tasks, along with the different data formats served by GeoServer.

Note: There are many more data sources available through *Extensions* and *Community modules*. If you are looking for a specific data format and not finding it here, please check those sections.

5.1 Data settings

This section describes how to manage load, manage, and publish data in the GeoServer web interface.

It describes the core configuration data types that GeoServer uses to access and publish geospatial information. Each subsection describes the data type pages which provide add, view, edit, and delete capabilities.

5.1.1 Layer Preview

This page provides layer views in various output formats. A layer must be enabled to be previewed.

Each layer row consists of a type, name, title, and available formats for viewing.

Field	Description
	Raster (grid)
	Polygon
	Line
	Point
	Other Geometry
	Layer Group
	Cascading WMS
	Unknown/Other

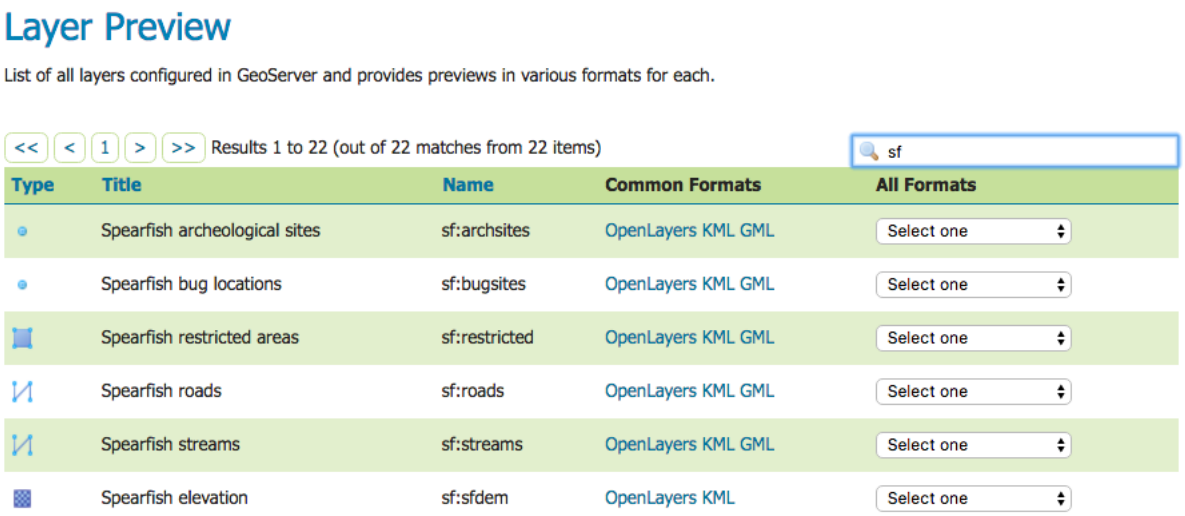


Fig. 5.1: Layer Preview Page

Name refers to the Workspace and Layer Name of a layer, while Title refers to the brief description configured in the *Edit Layer: Data* panel. In the following example, nurc refers to the Workspace, Arc_Sample refers to the Layer Name and "A sample ArcGrid field" is specified on the Edit Later Data panel.

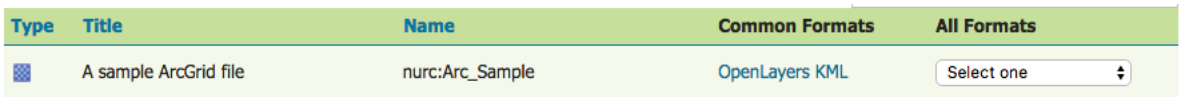


Fig. 5.2: Single Layer preview row

Output Formats

The Layer Preview page supports a variety of output formats for further use or data sharing. You can preview all three layer types in the common OpenLayers and KML formats. Similarly, using the "All formats" menu you can preview all layer types in seven additional output formats—AtomPub, GIF, GeoRss, JPEG, KML (compressed), PDF, PNG, SVG, and TIFF. Only Vector layers provide the WFS output previews, including the common GML as well as the CSV, GML3, GeoJSON and shapefile formats. The table below provides a brief description of all supported output formats, organized by output type (image, text, or data).

Image Outputs

All image outputs can be initiated from a WMS getMap request on either a raster, vector or coverage data. WMS are methods that allows visual display of spatial data without necessarily providing access to the features that comprise those data.

Format	Description
KML	KML (Keyhole Markup Language) is an XML-based language schema for expressing geographic data in an Earth browser, such as Google Earth or Google Maps. KML uses a tag-based structure with nested elements and attributes. For GeoServer, KML files are distributed as a KMZ, which is a zipped KML file.
JPEG	WMS output in raster format. The JPEG is a compressed graphic file format, with some loss of quality due to compression. It is best used for photos and not recommended for exact reproduction of data.
GIF	WMS output in raster format. The GIF (Graphics Interchange Format) is a bitmap image format best suited for sharp-edged line art with a limited number of colors. This takes advantage of the format's lossless compression, which favors flat areas of uniform color with well defined edges (in contrast to JPEG, which favors smooth gradients and softer images). GIF is limited to an 8-bit palette, or 256 colors.
SVG	WMS output in vector format. SVG (Scalable Vector Graphics) is a language for modeling two-dimensional graphics in XML. It differs from the GIF and JPEG in that it uses graphic objects rather than individual points.
TIFF	WMS output in raster format. TIFF (Tagged Image File Format) is a flexible, adaptable format for handling multiple data in a single file. GeoTIFF contains geographic data embedded as tags within the TIFF file.
PNG	WMS output in raster format. The PNG (Portable Network Graphics) file format was created as the free, open-source successor to the GIF. The PNG file format supports truecolor (16 million colors) while the GIF supports only 256 colors. The PNG file excels when the image has large, uniformly coloured areas.
OpenLayers	WMS GetMap request outputs a simple OpenLayers preview window. OpenLayers is an open source JavaScript library for displaying map data in web browsers. The OpenLayers output has some advanced filters that are not available when using a standalone version of OpenLayers. Further, the generated preview contains a header with easy configuration options for display. Version 3 of the OpenLayers library is used by default. Version 3 can be disabled with the *ENABLE_OL3* (true/false) format option or system property. For older browsers not supported by OpenLayers 3, version 2 is used regardless of the setting.
PDF	A PDF (Portable Document Format) encapsulates a complete description of a fixed-layout 2D document,including any text, fonts, raster images, and 2D vector graphics.

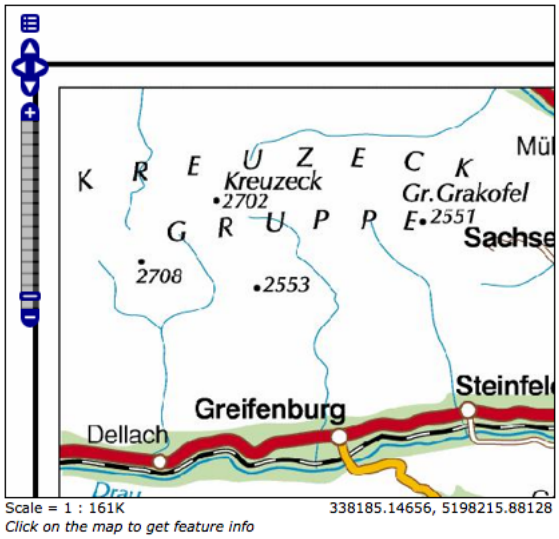


Fig. 5.3: Sample Image Output-an OpenLayers preview of nurc:Pk50095

Text Outputs

Format	Description
AtomPub	WMS output of spatial data in XML format. The AtomPub (Atom Publishing Protocol) is an application-level protocol for publishing and editing Web Resources using HTTP and XML. Developed as a replacement for the RSS family of standards for content syndication, Atom allows subscription of geo data.
GeoRss	WMS GetMap request output of vector data in XML format. RSS (Rich Site Summary) is an XML format for delivering regularly changing web content. GeoRss is a standard for encoding location as part of a RSS feed.supports Layers Preview produces a RSS 2.0 documents, with GeoRSS Simple geometries using Atom.
GeoJSON	JavaScript Object Notation (JSON) is a lightweight data-interchange format based on the JavaScript programming language. This makes it an ideal interchange format for browser based applications since it can be parsed directly and easily in to javascript. GeoJSON is a plain text output format that add geographic types to JSON.
CSV	WFS GetFeature output in comma-delimited text. CSV (Comma Separated Values) files are text files containing rows of data. Data values in each row are separated by commas. CSV files also contain a comma-separated header row explaining each row's value ordering. GeoServer's CSVs are fully streaming, with no limitation on the amount of data that can be outputted.

A fragment of a simple GeoRSS for nurc:Pk50095 using Atom:

```
<?xml version="1.0" encoding="UTF-8"?>
     <rss xmlns:atom="http://www.w3.org/2005/Atom"
          xmlns:georss="http://www.georss.org/georss" version="2.0">
        <channel>
              <title>Pk50095</title>
              <description>Feed auto-generated by GeoServer</description>
              <link>></link>
              <item>
                <title>fid--f04ca6b_1226f8d829e_-7ff4</title>
                <georss:polygon>46.722110379286 13.00635746384126
                     46.72697223230676 13.308182612644663 46.91359611878293
                     13.302316867622581 46.90870264238999 12.999446822650462
                     46.722110379286 13.00635746384126
                </georss:polygon>
                </item>
        </channel>
</rss>
```

Data Outputs

All data outputs are initiated from a WFS GetFeature request on vector data.

Format	Description
GML2/3	GML (Geography Markup Language) is the XML grammar defined by the Open Geospatial Consortium (OGC) to express geographical features. GML serves as a modeling language for geographic systems as well as an open interchange format for geographic data sharing. GML2 is the default (Common) output format, while GML3 is available from the "All Formats" menu.
Shapefile	The ESRI Shapefile, or simply a shapefile, is the most commonly used format for exchanging GIS data. GeoServer outputs shapefiles in zip format, with a directory of .cst, .dbf, .prg, .shp, and .shx files.

5.1.2 Workspaces

This section describes how to view and configure workspaces. Analogous to a namespace, a workspace is a container which organizes other items. In GeoServer, a workspace is often used to group similar layers together. Layers may be referred to by their workspace name, colon, layer name (for example `topp:states`). Two different layers can have the same name as long as they belong to different workspaces (for example `sf:states` and `topp:states`).

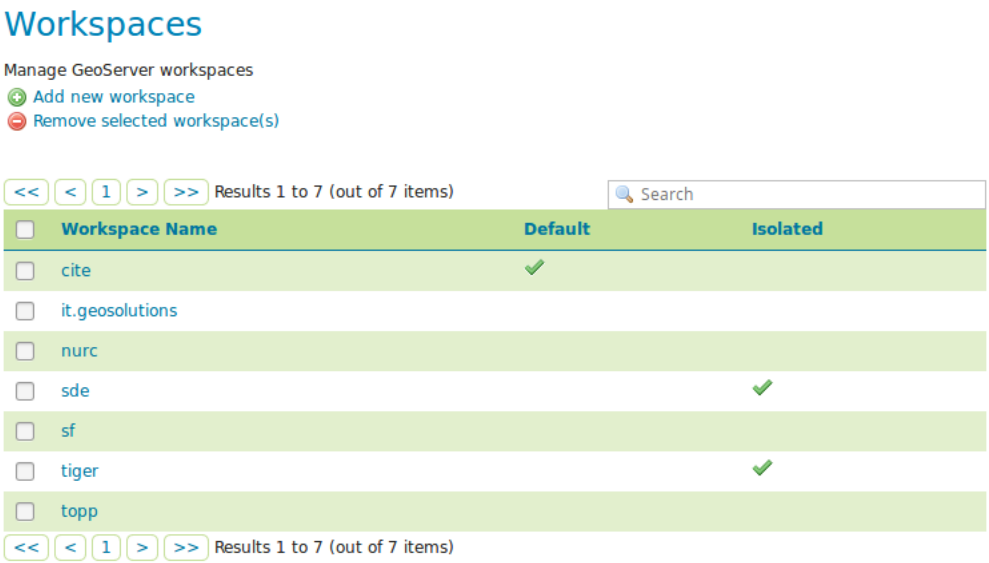


Fig. 5.4: Workspaces page

Edit a Workspace

To view or edit a workspace, click the workspace name. A workspace configuration page will be displayed.

A workspace is defined by a name and a Namespace URI (Uniform Resource Identifier). The workspace name is limited to ten characters and may not contain space. A URI is similar to a URL, except URIs do not need to point to a actual location on the web, and only need to be a unique identifier. For a Workspace URI, we recommend using a URL associated with your project, with perhaps a different trailing identifier. For example, `http://www.openplans.org/topp` is the URI for the "topp" workspace.

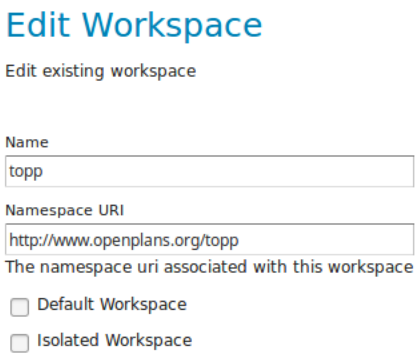


Fig. 5.5: Workspace named "topp"

Root Directory for REST PathMapper

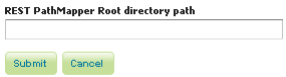


Fig. 5.6: Workspace Root Directory parameter

This parameter is used by the RESTful API as the *Root Directory* for uploaded files, following the structure:

```
${rootDirectory}/workspace/store[/<file>]
```

Note: This parameter is visible only when the **Enabled** parameter of the *Settings* section is checked.

Add a Workspace

The buttons for adding and removing a workspace can be found at the top of the Workspaces view page.

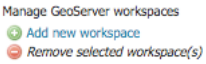


Fig. 5.7: Buttons to add and remove

To add a workspace, select the *Add new workspace* button. You will be prompted to enter the the workspace name and URI.

Remove a Workspace

To remove a workspace, select it by clicking the checkbox next to the workspace. Multiple workspaces can be selected, or all can be selected by clicking the checkbox in the header. Click the *Remove selected workspaces(s)* button. You will be asked to confirm or cancel the removal. Clicking *OK* removes the selected workspace(s).

Fig. 5.8: New Workspace page with example

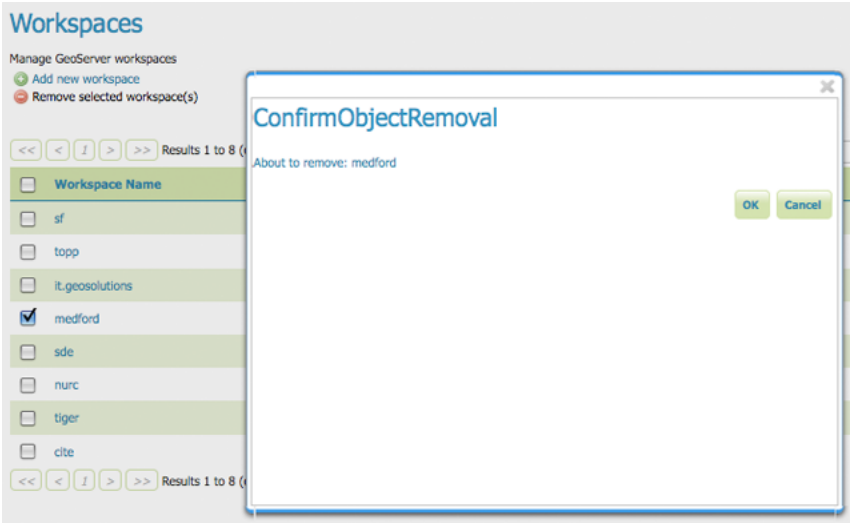


Fig. 5.9: Workspace removal confirmation

Isolated Workspaces

Isolated workspaces content is only visible and queryable in the context of a virtual service bound to the isolated workspace. This means that isolated workspaces content will not show up in global capabilities documents and global services cannot query isolated workspaces contents. Is worth mentioning that those restrictions don't apply to the REST API.

A workspace can be made isolated by checking the *Isolated Workspace* checkbox when creating or editing a workspace.

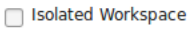


Fig. 5.10: Making a workspace isolated

An isolated workspace will be able to reuse a namespace already used by another workspace, but its resources (layers, styles, etc ...) can only be retrieved when using that workspace virtual services and will only show up in those virtual services capabilities documents.

It is only possible to create two or more workspaces with the same namespace in GeoServer if only one of them is non isolated, i.e. isolated workspaces have no restrictions in namespaces usage but two non isolated workspaces can't use the same namespace.

The following situation will be valid:

- Prefix: st1 Namespace: http://www.stations.org/1.0 Isolated: false
- Prefix: st2 Namespace: http://www.stations.org/1.0 Isolated: true
- Prefix: st3 Namespace: http://www.stations.org/1.0 Isolated: true

But not the following one:

- Prefix: st1 Namespace: http://www.stations.org/1.0 Isolated: false
- **Prefix: st2 Namespace: http://www.stations.org/1.0 Isolated: false**
- Prefix: st3 Namespace: http://www.stations.org/1.0 Isolated: true

At most only one non isolated workspace can use a certain namespace.

Consider the following image which shows to workspaces (st1 and st2) that use the same namespace (http://www.stations.org/1.0) and several layers contained by them:

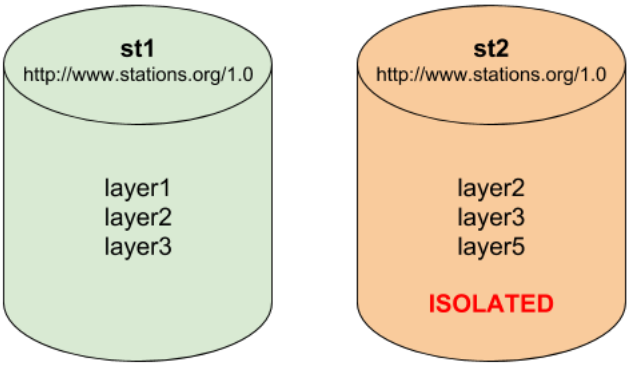


Fig. 5.11: Two workspaces using the same namespace, one of them is isolated.

In the example above st2 is the isolated workspace. Consider the following WFS GetFeature requests:

1. http://localhost:8080/geoserver/ows?service=WFS&version=2.0.0&request=DescribeFeatureType&typeName=layer2
2. http://localhost:8080/geoserver/st2/ows?service=WFS&version=2.0.0&request=DescribeFeatureType&typeName=layer2
3. http://localhost:8080/geoserver/ows?service=WFS&version=2.0.0&request=DescribeFeatureType&typeName=st1:layer2
4. http://localhost:8080/geoserver/st2/ows?service=WFS&version=2.0.0&request=DescribeFeatureType&typeName=st2:layer2
5. http://localhost:8080/geoserver/ows?service=WFS&version=2.0.0&request=DescribeFeatureType&typeName=st2:layer2
6. http://localhost:8080/geoserver/ows?service=WFS&version=2.0.0&request=DescribeFeatureType&typeName=layer5

The first request is targeting WFS global service and requesting layer2, this request will use layer2 contained by workspace st1. The second request is targeting st2 workspace WFS virtual service, layer2 belonging to workspace st2 will be used. Request three and four will use layer2 belonging to workspace, respectively, st1 and st2. The last two requests will fail saying that the feature type was not found, isolated workspaces content is not visible globally.

The rule of thumb is that resources (layers, styles, etc ...) belonging to an isolated workspace can only be retrieved when using that workspaces virtual services and will only show up in those virtual services capabilities documents.

5.1.3 Stores

A store connects to a data source that contains raster or vector data. A data source can be a file or group of files, a table in a database, a single raster file, or a directory (for example, a Vector Product Format library). The store construct allows connection parameters to be defined once, rather than for each dataset in a source. As such, it is necessary to register a store before configuring datasets within it.

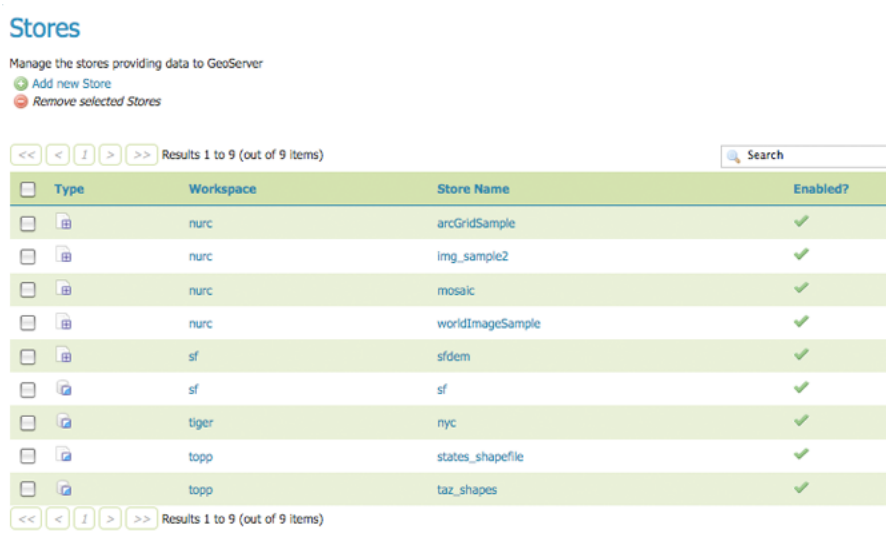


Fig. 5.12: Stores View

Store types

While there are many potential formats for data sources, there are only four kinds of stores. For raster data, a store can be a file. For vector data, a store can be a file, database, or server.

Type Icon	Description
	raster data in a file
	vector data in a file
	vector data in a database
	vector server (web feature server)

Edit a Store

To view or edit a store, click the store name. A store configuration page will be displayed. The exact contents of this page depend on the specific format of the store. See the sections *Vector data*, *Raster data*, and *Databases* for information about specific data formats. The example shows the configuration for the `nurc:ArcGridSample` store.

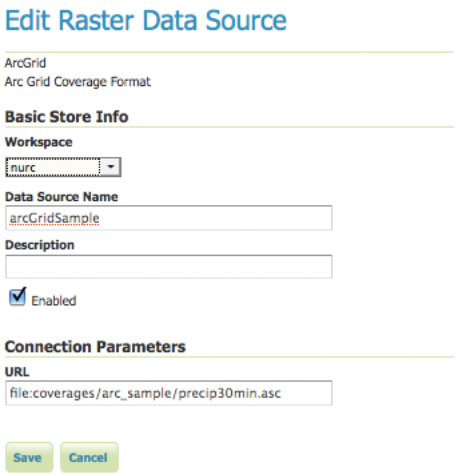


Fig. 5.13: Editing a raster data store

Basic Store Info

The basic information is common for all formats.

- **Workspace** - the store is assigned to the selected workspace
- **Data Source Name** - the store name as listed on the view page
- **Description** - (optional) a description that displays in the administration interface
- **Enabled** - enables or disables access to the store, along with all datasets defined for it

Connection Parameters

The connection parameters vary depending on data format.

Add a Store

The buttons for adding and removing a store can be found at the top of the Stores page.

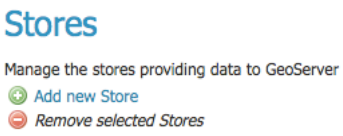


Fig. 5.14: Buttons to add and remove a Store

To add a store, select the *Add new Store* button. You will be prompted to choose a data source. GeoServer natively supports many formats (with more available via extensions). Click the appropriate data source to continue.

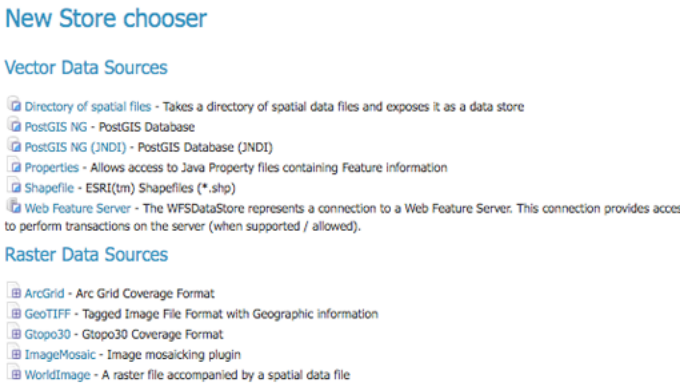


Fig. 5.15: Choosing the data source for a new store

The next page configures the store. Since connection parameters differ across data sources, the exact contents of this page depend on the store's specific format. See the sections *Vector data*, *Raster data*, and *Databases* for information on specific data formats. The example below shows the ArcGrid raster configuration page.

Remove a Store

To remove a store, click the checkbox next to the store. Multiple stores can be selected, or all can be selected by clicking the checkbox in the header.

Click the *Remove selected Stores* button. You will be asked to confirm the removal of the configuration for the store(s) and all resources defined under them. Clicking *OK* removes the selected store(s), and returns to the Stores page.

5.1.4 Layers

In GeoServer, the term "layer" refers to a raster or vector dataset that represents a collection of geographic features. Vector layers are analogous to "featureTypes" and raster layers are analogous to "coverages". All layers have a source of data, known as a Store. The layer is associated with the Workspace in which the Store is defined.

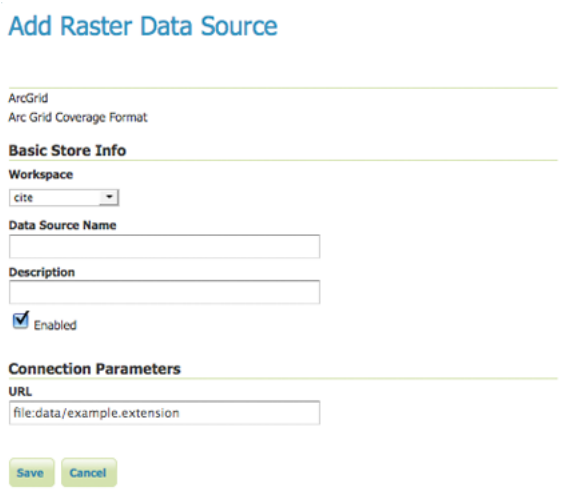


Fig. 5.16: Configuration page for an ArcGrid raster data source

Fig. 5.17: Stores selected for removal

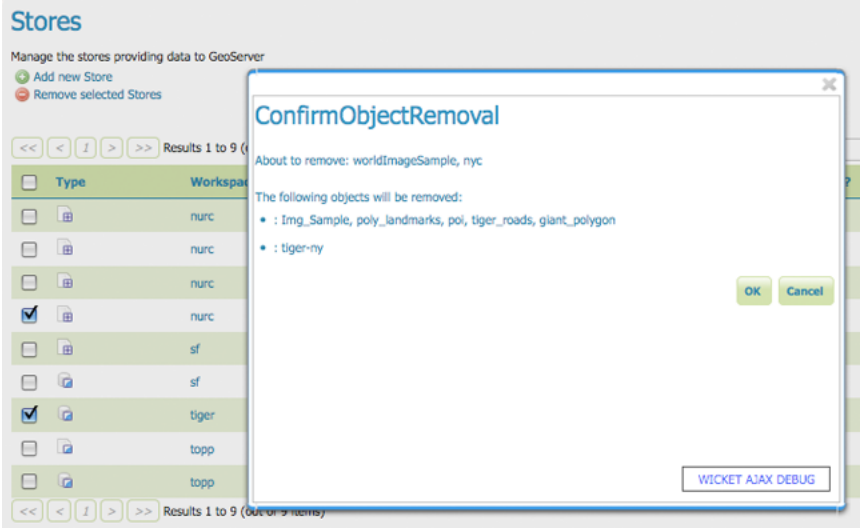


Fig. 5.18: Confirm removal of stores

In the Layers section of the web interface, you can view and edit existing layers, add (register) a new layer, or remove (unregister) a layer. The Layers View page displays the list of layers, and the Store and Workspace in which each layer is contained. The View page also displays the layer's status and native SRS.

Layer types

Layers can be divided into two types of data: raster and vector. These two formats differ in how they store spatial information. Vector types store information about feature types as mathematical paths—a point as a single x,y coordinate, lines as a series of x,y coordinates, and polygons as a series of x,y coordinates that start and end on the same place. Raster format data is a cell-based representation of features on the earth surface. Each cell has a distinct value, and all cells with the same value represent a specific feature.

Field	Description
	Raster (grid)
	Polygon
	Line
	Point

Add a Layer

At the upper left-hand corner of the layers view page there are two buttons for the adding and removal of layers. The green plus button allows you to add a new layer. The red minus button allows you to remove selected layers.

Clicking the *Add a new layer* button brings up a *New Layer Chooser* panel. The menu displays all currently enabled stores. From this menu, select the Store where the layer should be added.

Upon selection of a Store, a list is displayed of resources within the store. Resources which have already been published as layers are listed first, followed by other resources which are available to be published. In this example, `giant_polygon`, `poi`, `poly_landmarks` and `tiger_roads` are all existing layers within the NYC store.

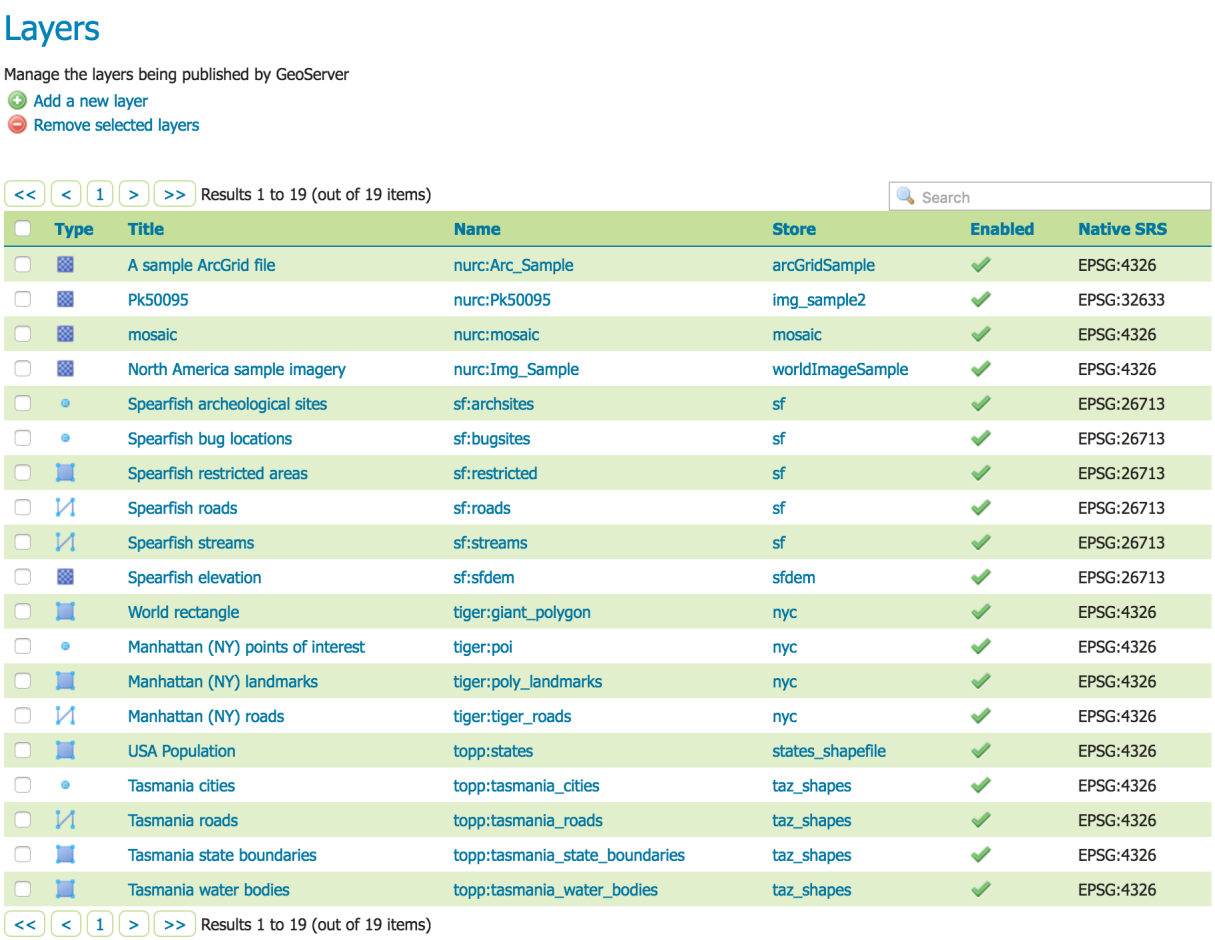


Fig. 5.19: Layers View

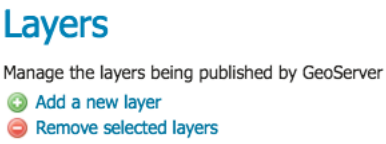


Fig. 5.20: Buttons to Add and Remove a Layer

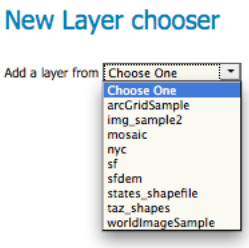


Fig. 5.21: List of all currently enabled stores

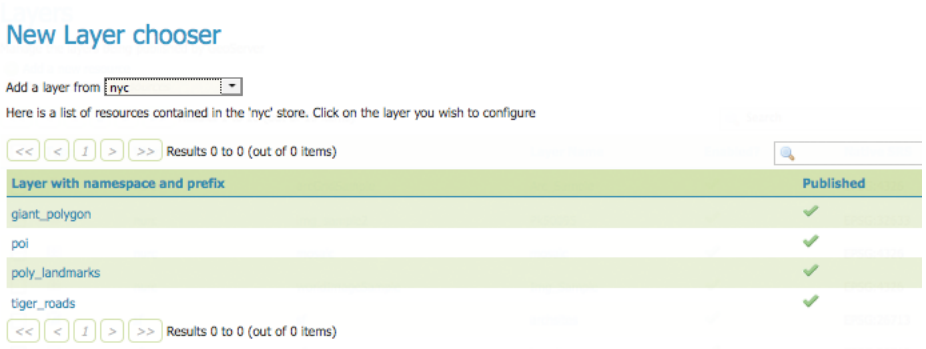


Fig. 5.22: List of published and available resources in a store

To add a layer for an available resource click *Publish*. To add a new layer for a published resource click *Publish Again*. (Note that when re-publishing the name of the new layer may have to be modified to avoid conflict with an existing layer.) The actions display an *Edit Layer* page to enter the definition of the new layer.

Remove a Layer

To remove a layer, select it by clicking the checkbox next to the layer. As shown below, multiple layers can be selected for batch removal. Note that selections for removal will not persist from one results pages to the next.

All layers can be selected for removal by clicking the checkbox in the header.

Once layer(s) are selected, the *Remove selected resources* link is activated. Once you've clicked the link, you will be asked to confirm or cancel the removal. Selecting *OK* removes the selected layer(s).

Edit Layer: Data

To view or edit a layer, click the layer name. A layer configuration page will be displayed. The *Data* tab, activated by default, allows you to define and change data parameters for a layer.

Basic Info

The beginning sections—Basic Resource Info, Keywords and Metadata link—are analogous to the *Service Metadata* section for WCS, WFS, and WMS. These sections provide "data about the data," specifically textual information that make the layer data easier to understand and work with. The metadata information will appear in the capabilities documents which refer to the layer.

- **Name**—Identifier used to reference the layer in WMS requests. (Note that for a new layer for an already-published resource, the name must be changed to avoid conflict.)
- **Title**—Human-readable description to briefly identify the layer to clients (required)
- **Abstract**—Describes the layer in detail
- **Keywords**—List of short words associated with the layer to assist catalog searching
- **Metadata Links**—Allows linking to external documents that describe the data layer. Currently only two standard format types are valid: TC211 and FGDC. TC211 refers to the metadata structure established by the ISO Technical Committee for Geographic Information/Geomatics (ISO/TC 211) while

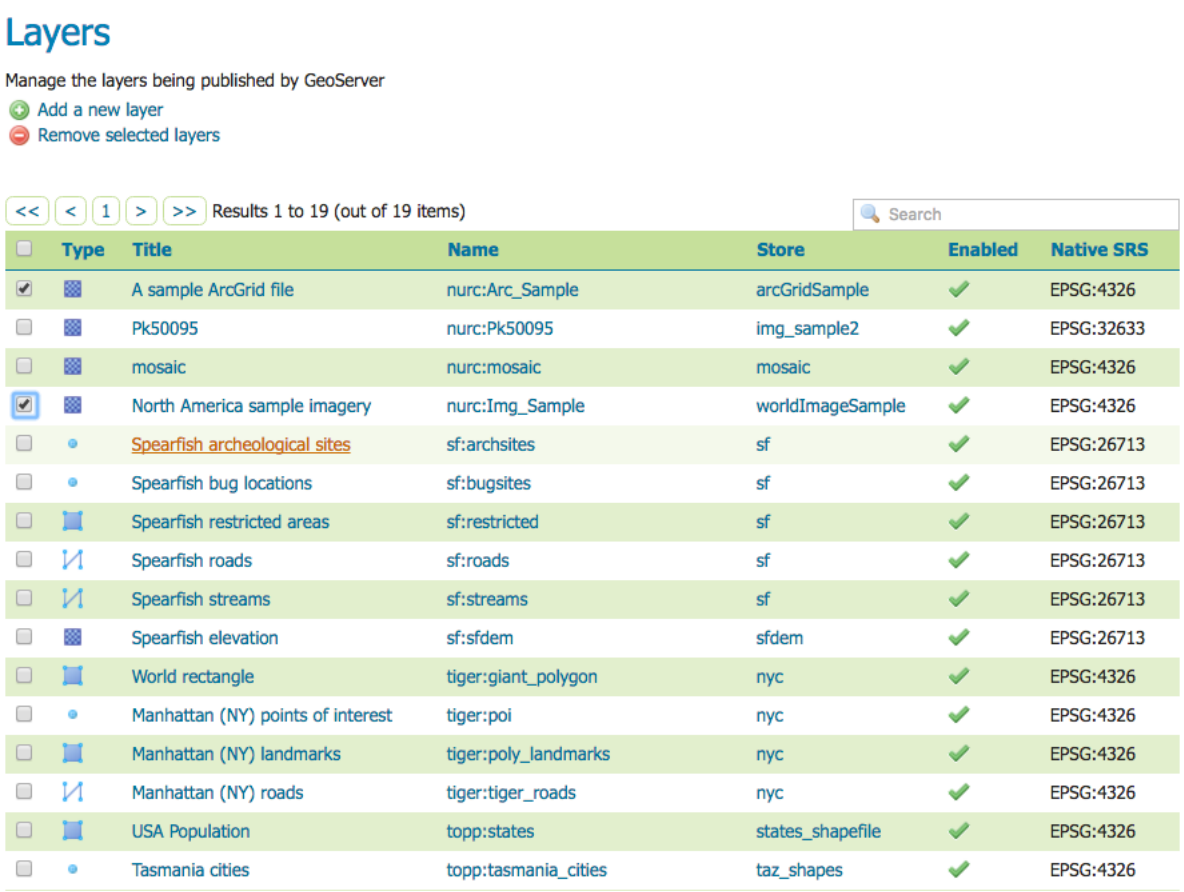


Fig. 5.23: Some layers selected for removal

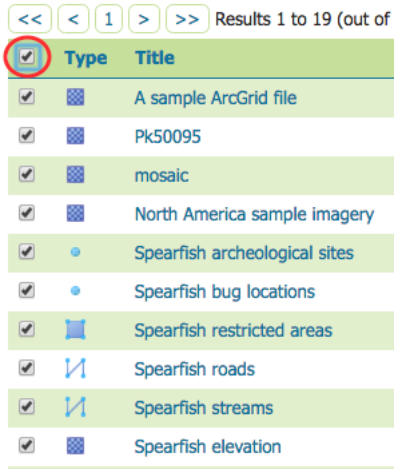


Fig. 5.24: All layers selected for removal

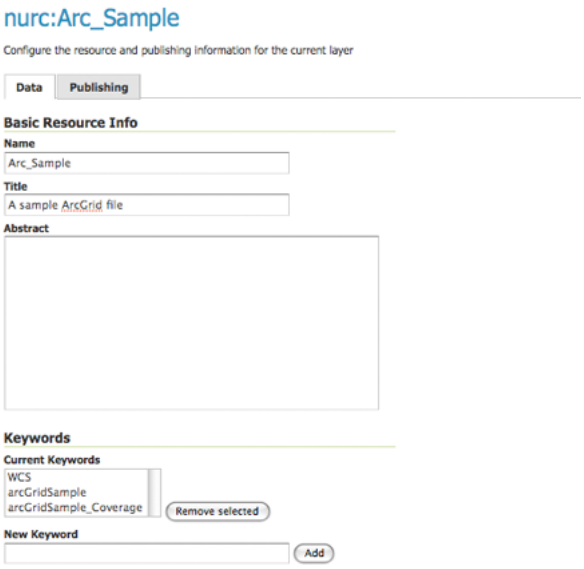


Fig. 5.25: Edit Layer: Data tab

FGDC refers to those set out by the Federal Geographic Data Committee (FGDC) of the United States.

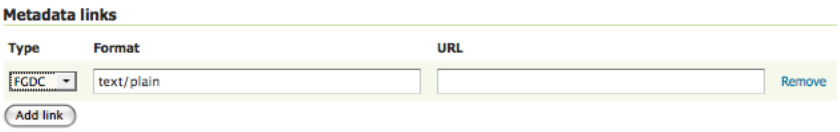


Fig. 5.26: Adding a metadata link in FGDC format

Coordinate Reference Systems

A coordinate reference system (CRS) defines how georeferenced spatial data relates to real locations on the Earth's surface. CRSes are part of a more general model called Spatial Reference Systems (SRS), which includes referencing by coordinates and geographic identifiers. GeoServer needs to know the Coordinate Reference System of your data. This information is used for computing the latitude/longitude bounding box and reprojecting the data during both WMS and WFS requests.

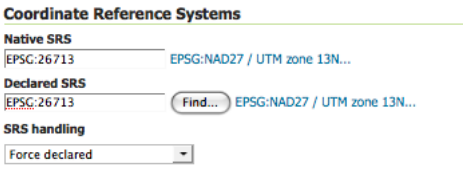


Fig. 5.27: Coordinate reference system of a layer

- **Native SRS**—Specifies the coordinate system the layer is stored in. Clicking the projection link displays a description of the SRS.

- **Declared SRS**—Specifies the coordinate system GeoServer publishes to clients
- **SRS Handling**—Determines how GeoServer should handle projection when the two SRSes differ. Possible values are:
  - **Force declared** (default): the declared SRS is forced upon the data, overwriting the native one. This is the default option and normally the best course of action, the declared code comes from the EPSG database and has a wealth of extra information in it, starting from a valid EPSG code, an area of validity, a link back in the database to find the best transformation steps to other coordinate reference systems should reprojection be required. Use this option when the source has no native CRS, has a wrong one, or has one matching the EPSG code (in order to get full metadata in the CRS used by GeoServer).
  - **Reproject from native**: This setting should be used when the native data set has a CRS that is not matching any official EPSG. OGC protocols need to advertise a EPSG code for the layers, with this setting the declared one will be advertised, and reprojection from native will happen on the fly as needed (in case a third CRS is requested, the reprojection will go directly from native to declared)
  - **Keep native**: this is a setting that should be used in very rare cases. Keeping native means using the declared one in the capabilities documents, but then using the native CRS in all othe requests (with no reprojection in between, unless explicitly requested from client). This is particularly problematic if the source is a shapefile, as the PRJ files lack all the extra information provided by the EPSG database (it will for example break WFS 1.1 and 2.0 SRS declarations in GML output). The setting meant to be used in cases where WMS is the primary target, and the native and declared CRSs have very small differences, avoiding on the fly reprojection and datum change.

In summary, use **Force Declared** as your primary option, **Reproject from native** only if your source data does not match any EPSG code, and **Keep Native** only if you really know what you're doing.

Bounding Boxes

The bounding box determines the extent of the data within a layer.

- **Native Bounding Box**—The bounds of the data specified in the Native SRS. These bounds can be generated by clicking the *Compute from data* button or they can be generated from the SRS definition by clicking the *Compute from SRS bounds* button. The SRS used depends on the *SRS Handling* chosen: the declared SRS when *Force declared* or *Reproject native to declared* are chosen, otherwise the native SRS is used. If the SRS does not have a bounding defined then none is generated.
- **Lat/Lon Bounding Box**—The bounds specified in geographic coordinates. These bounds can be calculated by clicking the *Compute from native bounds* button.

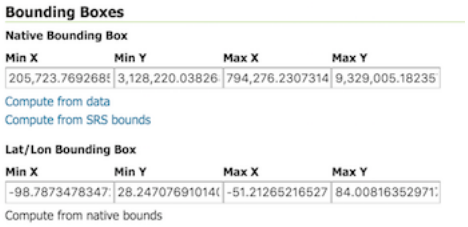


Fig. 5.28: Bounding Boxes of a layer

Coverage Parameters (Raster)

Optional coverage parameters are possible for certain types of raster data. For example, WorldImage formats request a valid range of grid coordinates in two dimensions known as a *ReadGridGeometry2D*. For ImageMosaic, you can use *InputImageThresholdValue*, *InputTransparentColor*, and *OutputTransparentColor* to control the rendering of the mosaic in terms of thresholding and transparency.

Curves support (Vector)

GeoServer can handle geometries containing circular arcs (initially only from Oracle Spatial and the "properties data store", though more data sources are planned).

These geometries are kept in memory in their circular representation for as long as possible, are properly visually depicted in WMS, and encoded in GML 3.x as curved.

There are two options pertaining the circular arcs:

- **Linear geometries can contain circular arcs** should be checked to inform the GML encoder that the layer can contain circular arcs among other linear segments in the geometries, and thus use "gml:Curve" in place of "gml:LineString" in GML 3.1 output format. This is required because there is no quick way to know from the data sources if the linear geometries do contain circular arcs, and the choice of top level GML elements influences whether it is possible, or not, to represent circular arcs in their natural form.
- **Linearization tolerance** controls how accurately the linearized version of geometries matches the original circular version of them. The tolerance can be expressed as an absolute number in the native unit of measure of the data, or it can be expressed in meters or feet using the "m" and "ft" suffixes (such as "10m" or "15ft").

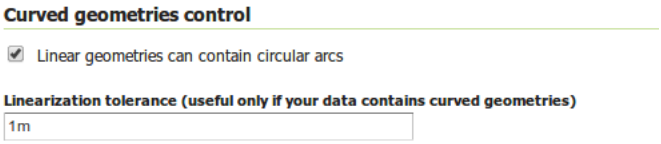


Fig. 5.29: Curved geometry control

Feature Type Details (Vector)

Vector layers have a list of the *Feature Type Details*. These include the *Property* and *Type* of a data source. For example, the `sf:archsites` layer shown below includes a geometry (`the_geom`) of type "point".

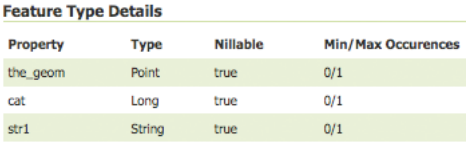


Fig. 5.30: Feature Type Details

The *Nillable* option refers to whether the property requires a value or may be flagged as being null. Meanwhile *Min/Max Occurrences* refers to how many values a field is allowed to have. Currently both *Nillable* and *Min/Max Occurrences* are set to `true` and `0/1` but may be extended with future work on complex features.

Restricting features showing up in the layer

By default GeoServer will publish all the features available in the layer. It is possible to restrict the features to a subset by specifying a CQL filter in the configuration:

Fig. 5.31: Restrict the features on layer by CQL filter

Note: It is recommended to use this setting for layers that are not meant to be edited. The filter is only applied to reads, if a WFS-T insert adds a feature not matching the filter, it will be added to the store anyways, but won't show up in any of the outputs.

Edit Layer: Publishing

The Publishing tab configures HTTP and WMS/WFS/WCS settings.

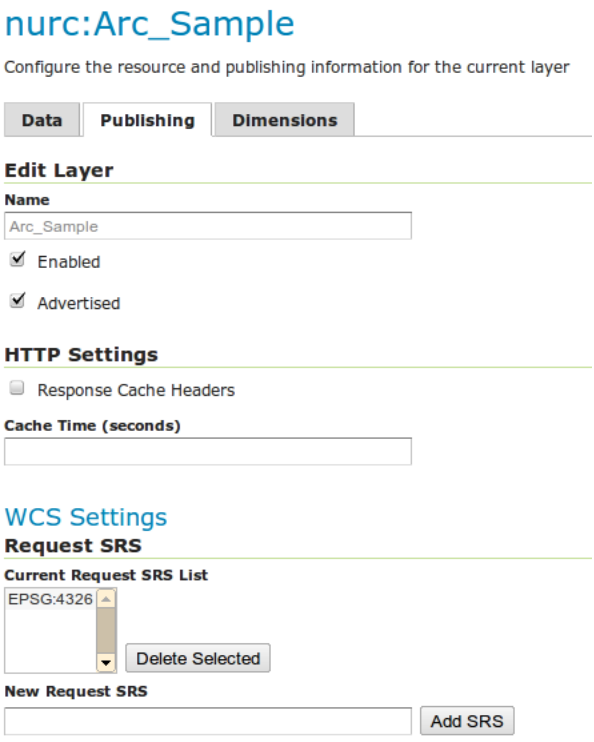


Fig. 5.32: Edit Layer: Publishing tab

- **Enabled**—A layer that is not enabled won't be available to any kind of request, it will just show up in the configuration (and in REST config)

- **Advertised**—A layer is advertised by default. A non-advertised layer will be available in all data access requests (for example, WMS GetMap, WMS GetFeature) but won't appear in any capabilities document or in the layer preview.

HTTP Settings

Cache parameters that apply to the HTTP response from client requests.

- **Response Cache Headers**— If selected, GeoServer will not request the same tile twice within the time specified in *Cache Time*. One hour measured in seconds (3600), is the default value for *Cache Time*.

Services Settings

Sets services configuration on layer level.

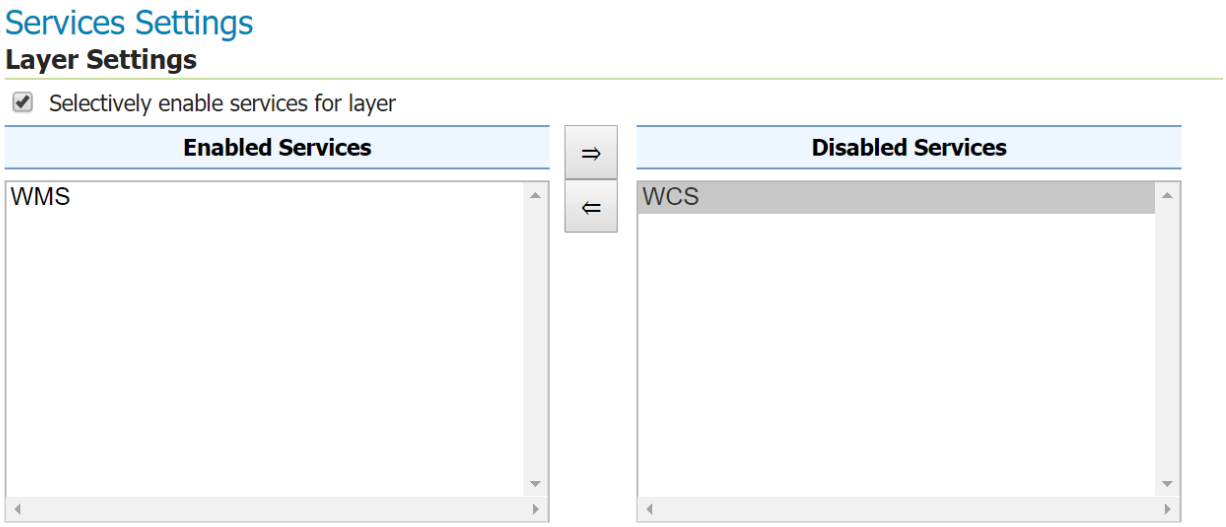


Fig. 5.33: Services Settings

- **Selectively enable services for layer**—Activate/deactivate service enable/disable configuration for the layer.
- **Enabled Services**—Selects enabled services list for this layer.
- **Disabled Services**—Selects disabled services list for this layer.

Note: It is also possible to set by-default disabled services to all layers using the `org.geoserver.service.disabled` system/env/servlet context variable. This variable accepts a comma separated list of services that should be disabled by default, in case the resource in question has no explicit configuration.

WMS Settings

Sets the WMS specific publishing parameters.

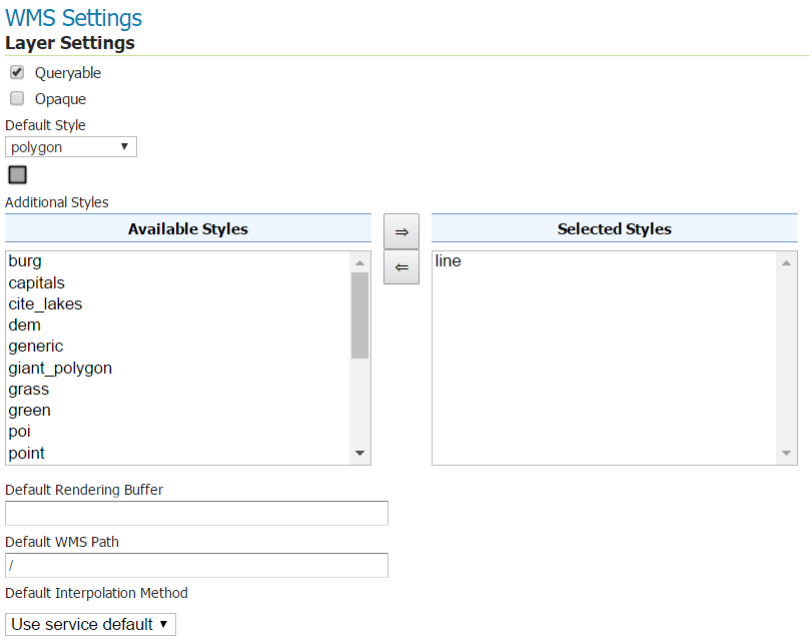


Fig. 5.34: WMS Settings

- **Queryable**—Controls whether the layer is queryable via WMS `GetFeatureInfo` requests.
- **Default style**—Style that will be used when the client does not specify a named style in GetMap requests.
- **Additional styles**—Other styles that can be associated with this layer. Some clients (and the GeoServer Layer Preview) will present those as styling alternatives for that layer to the user.
- **Default rendering buffer**—Default value of the `buffer` GetMap/GetFeatureInfo vendor parameter. See the *WMS vendor parameters* for more details.
- **Default WMS path**—Location of the layer in the WMS capabilities layer tree. Useful for building non-opaque layer groups
- **Default Interpolation Method**—Allows to specify a default resampling (interpolation) method for this layer. The available options are *Nearest neighbor*, *Bilinear*, *Bicubic*, or *Use service default*, which means that no layer specific configuration will be created (the default interpolation method selected in the WMS service configuration page will be used, see *Raster Rendering Options* for details). Can be overridden by the *interpolations vendor parameter*.

WMS Attribution

Sets publishing information about data providers.

- **Attribution Text**—Human-readable text describing the data provider. This might be used as the text for a hyperlink to the data provider's web site.

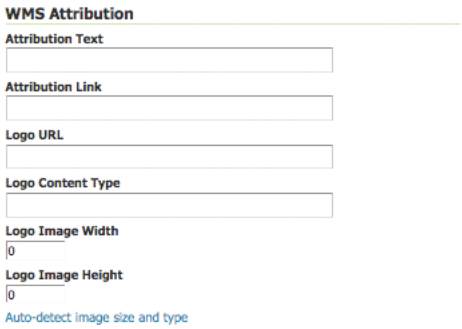


Fig. 5.35: WMS Attribution

- **Attribution Link**—URL to the data provider's website.
- **Logo URL**—URL to an image that serves as a logo for the data provider.
- **Logo Content Type, Width, and Height**—These fields provide information about the logo image that clients may use to assist with layout. GeoServer will auto-detect these values if you click the *Auto-detect image size and type* link at the bottom of the section. The text, link, and URL are each advertised in the WMS Capabilities document if they are provided. Some WMS clients will display this information to advise users which providers provide a particular dataset. If you omit some of the fields, those that are provided will be published and those that are not will be omitted from the Capabilities document.

WFS Settings

Sets the WFS specific publishing parameters.

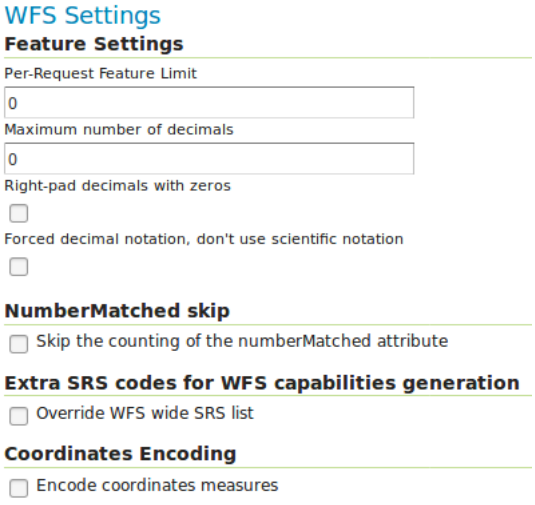


Fig. 5.36: WFS Settings

- **Per-Request Feature Limit**—Sets the maximum number of features for a layer a WFS GetFeature operation should generate (regardless of the actual number of query hits)
- **Maximum number of decimals**—Sets the maximum number of decimals in GML output.

Note: It is also possible to override the `OtherSRS/OtherCRS` list configured in the WFS service, including overriding it with an empty list if need be. The input area will accept a comma separated list of EPSG codes:

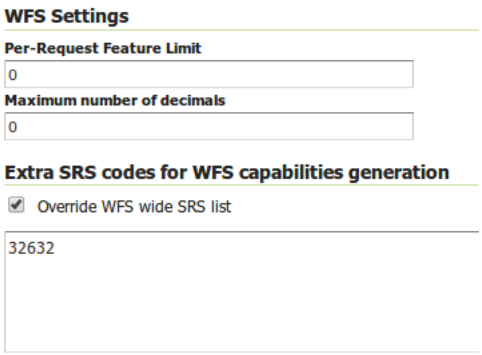


Fig. 5.37: WFS otherSRS/otherCRS override

The list will be used only for the capabilities document generation, but will not be used to limit the actual target SRS usage in GetFeature requests.

- **Encode coordinates measures**—Checking this setting will cause coordinates measures ("M") to be encoded in WFS output formats that support measures. The default (not checked) is to not encode coordinates measures.

WCS Settings

- **Request SRS**—Provides a list of SRSs the layer can be converted to. *New Request SRS* allows you to add an SRS to that list.
- **Interpolation Methods**—Sets the raster rendering process, if applicable.
- **Formats**—Lists which output formats a layers supports.
- **GeoSearch**—When enabled, allows the Google Geosearch crawler to index from this particular layer. See What is a Geo Sitemap? for more information.

KML Format Settings

Limits features based on certain criteria, otherwise known as **regionation**.

- **Default Regionating Attribute**—Choose which feature should show up more prominently than others.
- **Regionating Methods**—There are four types of regionating methods:
  - *external-sorting*—Creates a temporary auxiliary database within GeoServer. The first request to build an index takes longer than subsequent requests.
  - *geometry*—Externally sorts by length (if lines) or area (if polygons)
  - *native-sorting*—Uses the default sorting algorithm of the backend where the data is hosted. It is faster than external-sorting, but will only work with PostGIS datastores.

- *random*—Uses the existing order of the data and does not sort

Edit Layer: Dimensions

GeoServer supports adding specific dimensions to WMS layers, as specified in WMS 1.1.1 and WMS 1.3.0 standards. There are two pre-defined dimensions in the WMS standards mentioned above, **TIME** and **ELEVATION**. Enabling dimensions for a layer allows users to specify those as extra parameters in GetMap requests, useful for creating maps or animations from underlying multi-dimensional data.

These dimensions can be enabled and configured on the Dimensions tab.

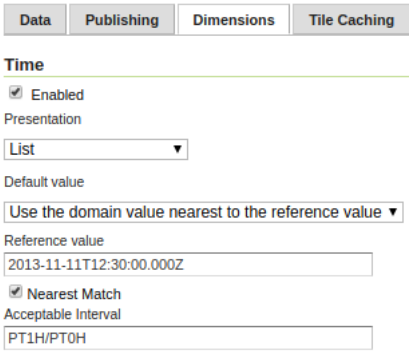


Fig. 5.38: TIME dimension enabled for a WMS layer

For each enabled dimension the following configuration options are available:

- **Attribute**—Attribute name for picking the value for this dimension (vector only). This is treated at start of the range if **End attribute** is also given.
- **End attribute**—Attribute name for picking the end of the value range for this dimension (optional, vector only).
- **Presentation**—The presentation type for the available values in the capabilities document. Either *each value separately (list)*, *interval and resolution*, or *continuous interval*.
- **Default value**—Default value to use for this dimension if none is provided with the request. Select one of from four strategies:
  - **smallest domain value**—Uses the smallest available value from the data
  - **biggest domain value**—Uses the biggest available value from the data
  - **nearest to the reference value**—Selects the data value closest to the given reference value
  - **reference value**—Tries to use the given reference value as-is, regardless of whether its actually available in the data or not.
- **Reference value**—The default value specifier. Only shown for the default value strategies where its used.
- **Nearest match**—Whether to enable, or not, WMS nearest match support on this dimension. Currently supported only on the time dimension.
- **Acceptable interval**—A maximum search distance from the specified value (available only when nearest match is enabled). Can be empty (no limit), a single value (symmetric search) or using a `before/after` syntax to specify an asymmetric search range. Time distances should specified using

the ISO period syntax. For example, `PT1H/PT0H` allows to search up to one hour before the user specified value, but not after.

For time dimension the value must be in ISO 8601 DateTime format `yyyy-MM-ddThh:mm:ss.SSSZ` For elevation dimension, the value must be and integer of floating point number.

Only for the "Reference value" strategy, it is also possible to use ranges or times and ranges of elevation, in the form `fromValue/toValue`. Only for the "Reference value" strategy, and limited to times, it's also possible to use relative times like `P1M/PRESENT`, but caution is given that the reference value is copied verbatim into the capabilities document, and as a result, not all client might be recognizing that syntax.

Note: For more information on specifying times, please see the section on *Time Support in GeoServer WMS*.

5.1.5 Layer Groups

A layer group is a container in which layers and other layer groups can be organized in a hierarchical structure. A layer group can be referred to by a single name in WMS requests. This allows simpler requests, as one layer can be specified instead of multiple individual layers. A layer group also provides a consistent, fixed ordering of the layers it contains, and can specify alternate (non-default) styles for layers.

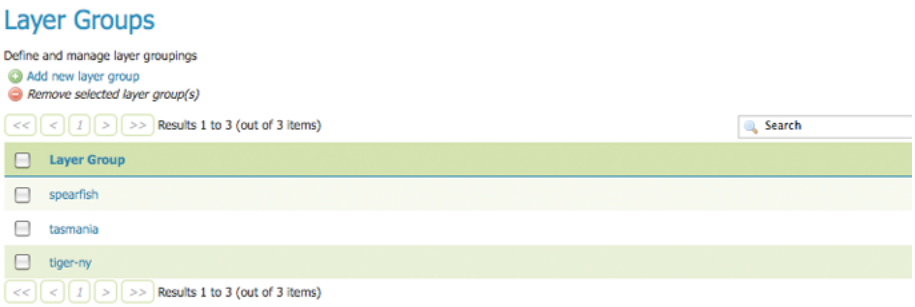


Fig. 5.39: Layer Groups page

Layer Group modes

Layer group behaviour can be configured by setting its *mode*. There are 5 available values:

- **single**: the layer group is exposed as a single layer with a name, acting as an alias for a list of layers. The layers are still showing up as top level entries in the WMS capabilities document (unless explicitly referred by a tree group).
- **opaque container**: the layer group is exposed as a single layer with a name, acting as an alias for a list of layers. However, the layers and sub-groups contained in it won't show up in the capabilities document (unless explicitly referred by a tree group) and won't be available by themselves in WMS calls and in the WMS capabilities document, but only as part of the group.
- **named tree**: the layer group can be referred to by one name, but also exposes its nested layers and groups in the capabilities document.
- **container tree**: the layer group is exposed in the capabilities document, but does not have a name, making it impossible to render it on its own. This is called "containing category" in the WMS specification.

- **Earth Observation tree**: a special type of group created to manage the WMS Earth Observation requirements. This group does not render its nested layers and groups, but only a "preview layer" called Root Layer. When this mode is chosen, a new field "Root Layer" will be exposed in the configuration UI.

If a layer is included in any non *single* mode group, it will no longer be listed in the flat layer list. It will still be possible to include the layer in other layer groups.

Layer Group Mode	Named	Contains Children	Lists Children	Details
Single	named		no	
Opaque Container	named	yes	no	hides children
Named Tree	named	yes	lists children	
Container Tree		yes	lists children	
Earth Observation Tree	named	yes	lists children	has root layer

Edit a Layer Group

To view or edit a layer group, click the layer group name. A layer group configuration page will be displayed. The initial fields allow you to configure the name, title, abstract, workspace, bounds, projection and mode of the layer group. To automatically set the bounding box, select the *Generate Bounds* button or the *Generate Bounds From CRS* button to use the bounds defined in the CRS (if available). You may also provide your own custom bounding box extents. To select an appropriate projection click the *Find* button.

Note: A layer group can contain layers with dissimilar bounds and projections. GeoServer automatically reprojects all layers to the projection of the layer group.

The table at the bottom of the page lists layers and groups contained within the current layer group. We refer to layers and layer groups as *publishable elements*. When a layer group is processed, the layers are rendered in the order provided, so the *publishable elements* at the bottom of list will be rendered last and will show on top of the other *publishable elements*.

A *publishable element* can be positioned higher or lower on this list by clicking the green up or down arrows, respectively, or can be simply dragged in the target position. The layer at the top of the list is the first one to be painted, the layer below it will be painted second, and so on, the last layer will be painted on top of all others (this is the so called "painter's model").

The *Style* column shows the style associated with each layer. To change the style associated with a layer, click the appropriate style link. A list of enabled styles will be displayed. Clicking on a style name reassigns the layer's style.

To remove a *publishable element* from the layer group, select its button in the *Remove* column. You will now be prompted to confirm or cancel this deletion.

A layer can be added to the list by clicking the *Add Layer...* button at the top of the table. From the list of layers, select the layer to be added by clicking the layer name. The selected layer will be appended to the bottom of the *publishable* list.

A layer group can be added by clicking the *Add Layer Group...* button at the top of the table. From the list of layer groups, select the layer group to be added by clicking its name. The selected group will be appended to the bottom of the *publishable* list.

A style group can be added by clicking the *Add Style Group...* button at the top of the table. From the list of styles, select the *style group* to be added by clicking its name. The selected style will be appended to the bottom of the *publishable* list.

You can view layer groups in the *Layer Preview* section of the web admin.

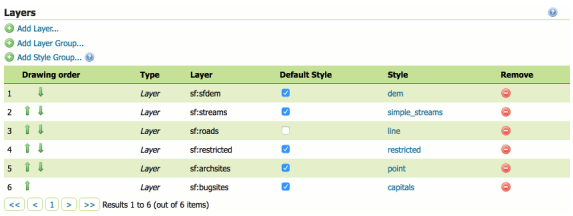


Fig. 5.40: Layer Groups Edit page

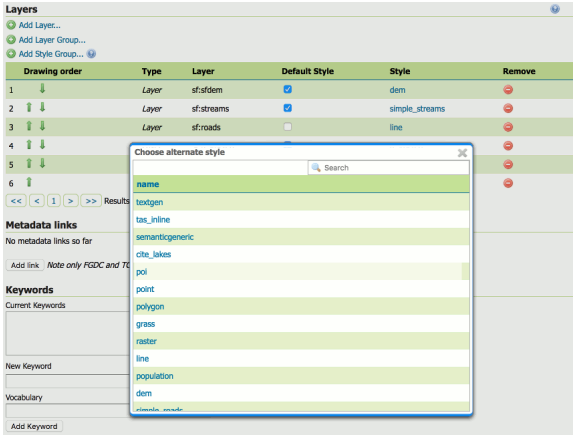


Fig. 5.41: Style editing for a layer within a layer group

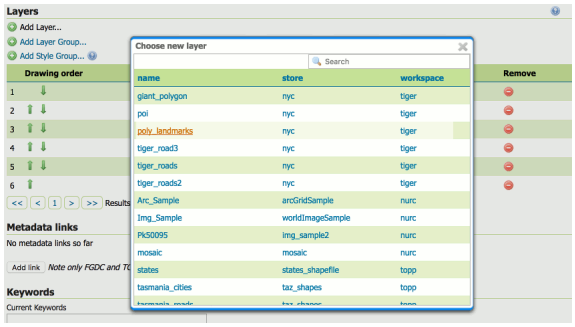


Fig. 5.42: Dialog for adding a layer to a layer group

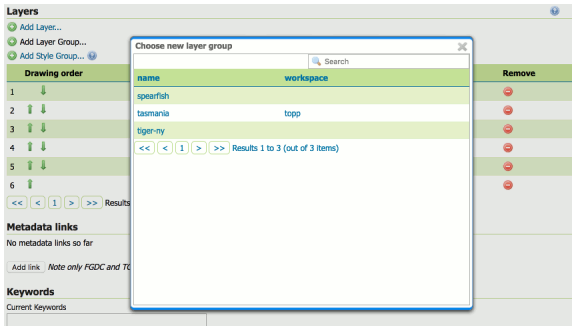


Fig. 5.43: Dialog for adding a layer group to a layer group

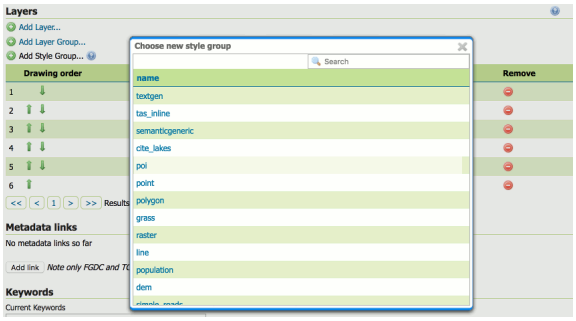


Fig. 5.44: Dialog for adding a style group to a layer group

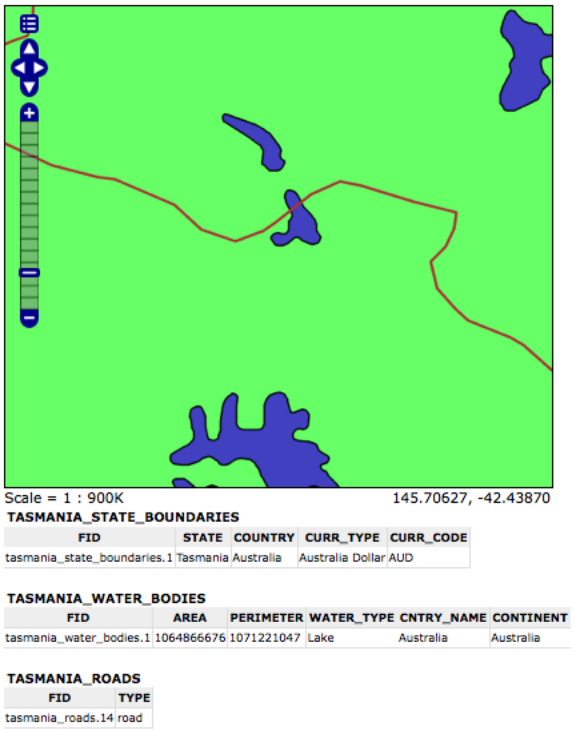


Fig. 5.45: Openlayers preview of the layer group "tasmania"

Note: By default, a layer group is queryable when at least a child layer is queryable. Uncheck "Queryable" box if you want to explicitly indicate that it is not queryable independently of how the child layers are configured.

Add a Layer Group

The buttons for adding and removing a layer group can be found at the top of the *Layer Groups* page.

Fig. 5.46: Buttons to add or remove a layer group

To add a new layer group, select the "Add a new layer group" button. You will be prompted to name the layer group.

When finished, click *Submit*. You will be redirected to an empty layer group configuration page. Begin by adding layers by clicking the *Add layer...* button (described in the previous section). Once the layers are positioned accordingly, press *Generate Bounds* to automatically generate the bounding box and projection. You may also press the *Generate Bounds From CRS* button to use the CRS bounds (if available). Press *Save* to save the new layer group.

Fig. 5.47: New layer group dialog

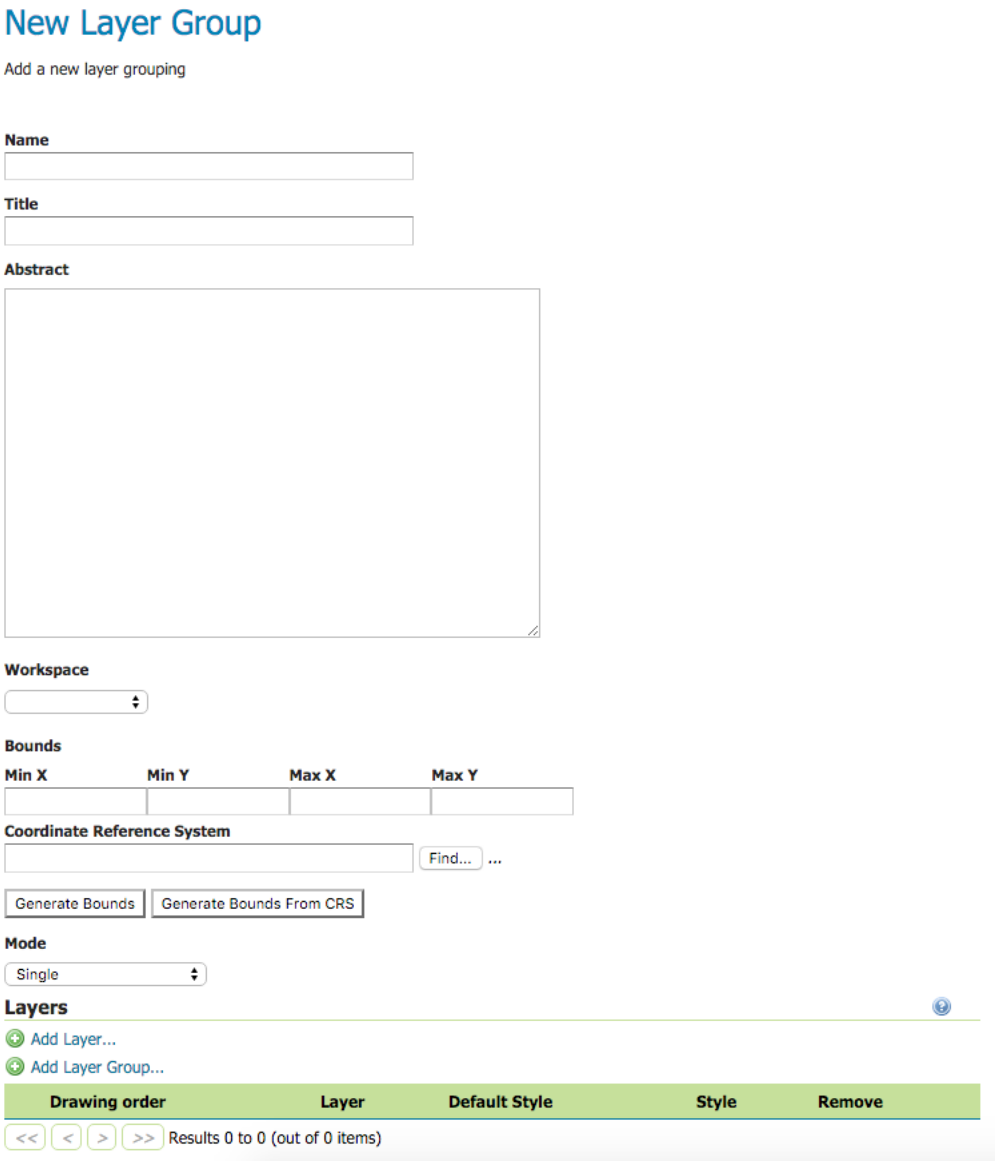


Fig. 5.48: New layer group configuration page

Remove a Layer Group

To remove a layer group, select it by clicking the checkbox next to the layer group. Multiple layer groups can be selected, or all can be selected by clicking the checkbox in the header. Click the *Remove selected layer group(s)* link. You will be asked to confirm or cancel the deletion. Selecting *OK* removes the selected layer group(s).

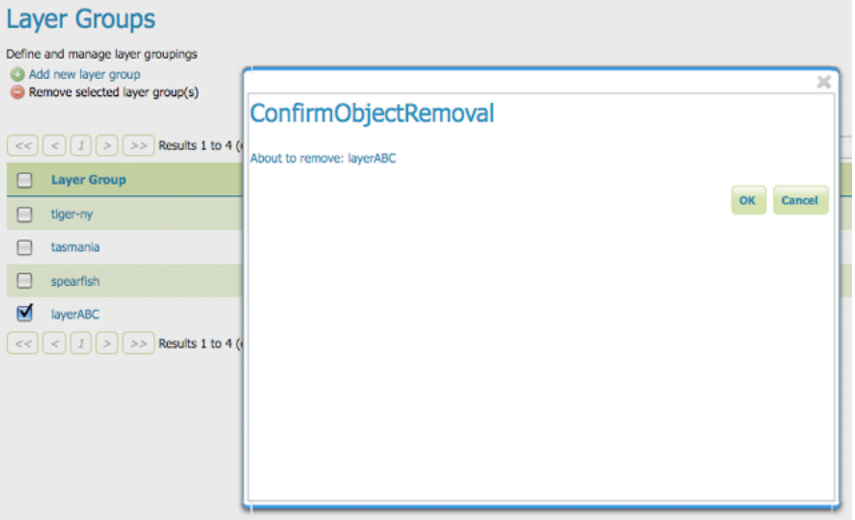


Fig. 5.49: Removing a layer group

Layer Group Keywords

Is possible to associate a layer group with some keywords that will be used to assist catalog searching.

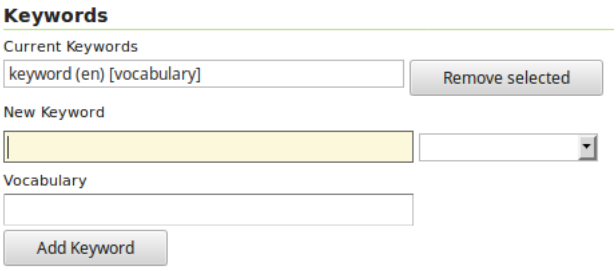


Fig. 5.50: Layer groups keywords editor

Layer groups keywords will no be merged with contained layers keywords but keywords of a layer group should be logically inherited by contained layers.

Note: Information on the Styles pages can be found on the *Styles* page.

5.2 Vector data

This section discusses the vector data sources that GeoServer can access.

The standard GeoServer installation supports the loading and serving of the following data formats:

5.2.1 Shapefile

A shapefile is a popular geospatial vector data format.

Note: While GeoServer has robust support for the shapefile format, it is not the recommended format of choice in a production environment. Databases such as PostGIS are more suitable in production and offer better performance and scalability. See the section on *Running in a production environment* for more information.

Adding a shapefile

A shapefile is actually a collection of files (with the extensions: `.shp`, `.dbf`, `.shx`, `.prj`, and sometimes others). All of these files need to be present in the same directory in order for GeoServer to accurately read them. As with all formats, adding a shapefile to GeoServer involves adding a new store to the existing *Stores* through the *Web administration interface*.

> **Warning:** The `.prj` file, while not mandatory, is strongly recommended when working with GeoServer as it contains valuable projection info. GeoServer may not be able to load your shapefile without it!

To begin, navigate to *Stores* → *Add a new store* → *Shapefile*.

Option	Description
Workspace	Name of the workspace to contain the store. This will also be the prefix of the layer created from the store.
Data Source Name	Name of the shapefile as known to GeoServer. Can be different from the filename. The combination of the workspace name and this name will be the full layer name (ex: topp:states).
Description	Description of the shapefile/store.
Enabled	Enables the store. If unchecked, no data in the shapefile will be served.
URL	Location of the shapefile. Can be an absolute path (such as `file:C:\Data\shapefile.shp`) or a path relative to the data directory (such as `file:data/shapefile.shp`.
namespace	Namespace to be associated with the shapefile. This field is altered by changing the workspace name.
create spatial index	Enables the automatic creation of a spatial index.
charset	Character set used to decode strings from the `.dbf` file.
memory mapped buffer Cache and reuse memory maps	Enables the use of memory mapped I/O, improving caching of the file in memory. **Turn off on Windows servers**.

When finished, click *Save*.

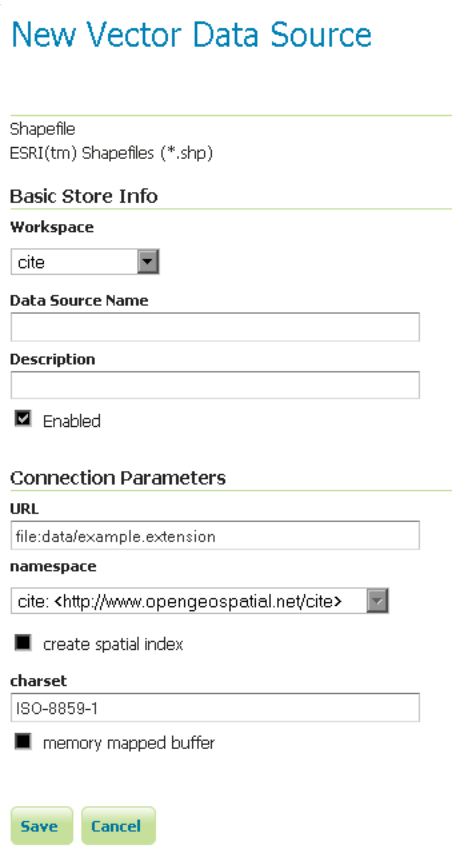


Fig. 5.51: *Adding a shapefile as a store*

Configuring a shapefile layer

Shapefiles contain exactly one layer, which needs to be added as a new layer before it will be able to be served by GeoServer. See the section on *Layers* for how to add and edit a new layer.

5.2.2 Directory of spatial files

The directory store automates the process of loading multiple shapefiles into GeoServer. Loading a directory that contains multiple shapefiles will automatically add each shapefile to GeoServer.

Note: While GeoServer has robust support for the shapefile format, it is not the recommended format of choice in a production environment. Databases such as PostGIS are more suitable in production and offer better performance and scalability. See the section on *Running in a production environment* for more information.

Adding a directory

To begin, navigate to *Stores* → *Add a new store* → *Directory of spatial files*.

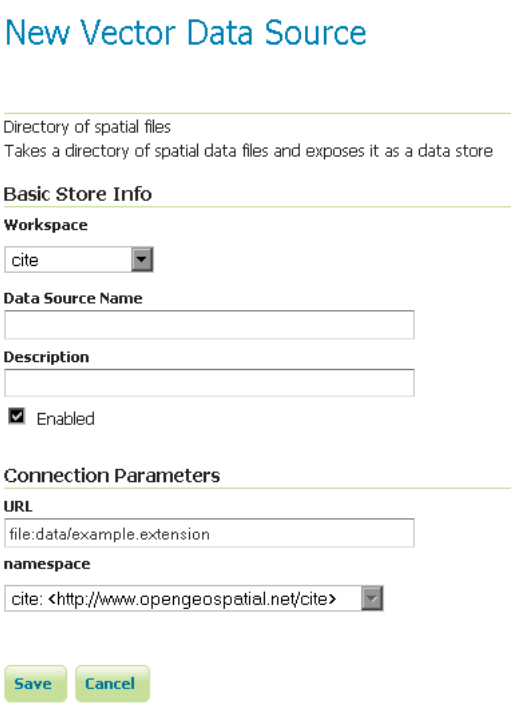


Fig. 5.52: *Adding a directory of spatial files as a store*

Option	Description
Workspace	Name of the workspace to contain the store. This will also be the prefix of all of the layer names created from shapefiles in the store.
Data Source Name	Name of the store as known to GeoServer.
Description	Description of the directory store.
Enabled	Enables the store. If disabled, no data in any of the shapefiles will be served.
URL	Location of the directory. Can be an absolute path (such as `file:C:\Data\shapefile_directory`) or a path relative to the data directory (such as `file:data/shapefile_directory`.
namespace	Namespace to be associated with the store. This field is altered by changing the workspace name.

When finished, click *Save*.

Configuring shapefiles

All of the shapefiles contained in the directory store will be loaded as part of the directory store, but they will need to be individually configured as new layers they can be served by GeoServer. See the section on *Layers* for how to add and edit new layers.

5.2.3 Java Properties

The Properties data store provides access to one or more feature types (layers) stored in Java property files; these are plain text files stored on the local filesystem. The Properties data store was never intended to be shipped with GeoServer. It originated in a GeoTools tutorial, and later found widespread use by developers in automated tests that required a convenient store for small snippets of data. It slipped into GeoServer through the completeness of the packaging process, and was automatically detected and offered to users via the web interface. The Property data store has proved useful in tutorials and examples.

- We do not recommend the use the Properties data store for large amounts of data, with either many features or large geometries. Its performance will be terrible.
- For small data sets, such as collections of a few dozen points, you may find it to be satisfactory. For example, if you have a few points you wish to add as an extra layer, and no convenient database in which store them, the Properties data store provides a straightforward means of delivering them.
- Changes to a property file are immediately reflected in GeoServer responses. There is no need to recreate the data store unless the first line of a property file is changed, or property files are added or removed.

Adding a Properties data store

By default, *Properties* will be an option in the *Vector Data Sources* list when creating a new data store.

Fig. 5.53: *Properties in the list of vector data stores*

Configuring a Properties data store

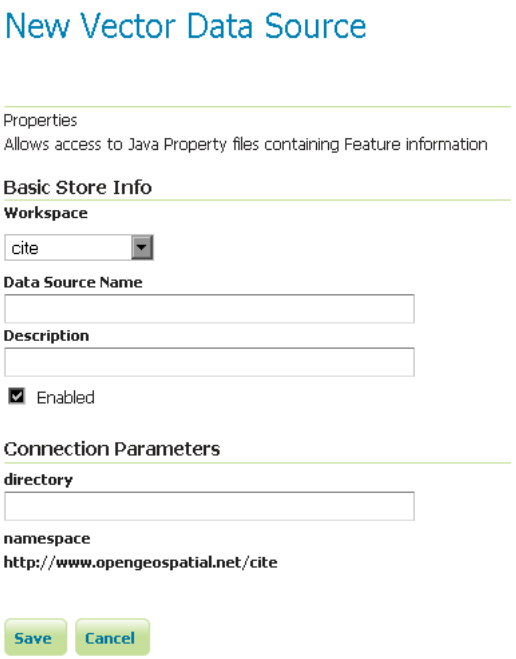


Fig. 5.54: *Configuring a Properties data store*

Option	Description
`Workspace`	Sets the namespace prefix of the feature types (layers) and their properties
`Data Source Name`	Unique identifier to distinguish this data store
`Description`	Optional text giving a verbose description of the data store
`Enabled`	Features will be delivered only if this option is checked
`directory`	Filesystem path to a directory containing one or more property files, for example `/usr/local/geoserver/data/ex`

Every property file `TYPENAME.properties` in the designated directory is served as a feature type `TYPENAME` (the name of the file without the `.properties`), in the namespace of the data store.

Before a feature type (layer) can be used, you must edit it to ensure that its bounding box and other metadata is configured.

Property file format

The property file format is a subset of the Java properties format: a list of lines of the form `KEY=VALUE`.

This example `stations.properties` defines four features of the feature type (layer) `stations`:

```
_=id:Integer,code:String,name:String,location:Geometry:srid=4326
stations.27=27|ALIC|Alice Springs|POINT(133.8855 -23.6701)
stations.4=4|NORF|Norfolk Island|POINT(167.9388 -29.0434)
stations.12=12|COCO|Cocos|POINT(96.8339 -12.1883)
stations.31=31|ALBY|Albany|POINT(117.8102 -34.9502)
```

- Blank lines are not permitted anywhere in the file.
- The first line of the property file begins with _= and defines the type information required to interpret the following lines.
  - Comma separated values are of the form `NAME:TYPE`
  - Names are the property name that are used to encode the property in WFS responses.
  - Types include `Integer`, `String`, `Float`, and `Geometry`
  - `Geometry` can have an extra suffix `:srid=XXXX` that defines the Spatial Reference System by its numeric EPSG code. Note that geometries defined in this way are in longitude/latitude order.
- Subsequent lines define features, one per line.
  - The key before the = is the feature ID (`fid` or `gml:id` in WFS responses). Each must be an NCName.
  - Feature data follows the = separated by vertical bars (`|`). The types of the data must match the declaration on the first line.
  - Leave a field empty if you want it to be null; in this case the property will be ignored.

Note that in this example `srid=4326` sets the spatial reference system (SRS) to `EPSG:4326`, which is by convention in longitude/latitude order when referred to in the short form. If you request these features in GML 3 you will see that GeoServer correctly translates the geometry to the URN form SRS `urn:x-ogc:def:crs:EPSG:4326` in latitude/longitude form. See the *WFS settings* page for more on SRS axis order options.

5.2.4 GeoPackage

GeoPackage is an SQLite based standard format that is able to hold multiple vector and raster data layers in a single file.

GeoPackage files can be used both as Raster Data Stores as well as Vector Data Stores (so that both kinds of layers can published).

Adding a GeoPackage Vector Data Store

When the extension has been installed, *GeoPackage* will be an option in the *Vector Data Sources* list when creating a new data store.

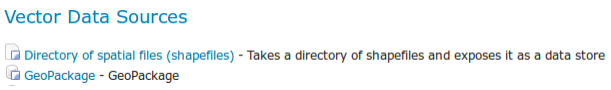


Fig. 5.55: *GeoPackage in the list of vector data stores*

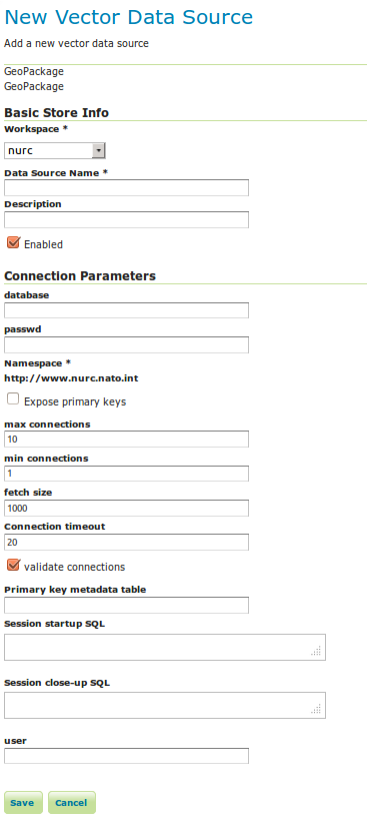


Fig. 5.56: *Configuring a GeoPackage Vector data store*

Option	Description
database	URI specifying geopackage file.
user	User to access database.
passwd	Password to access database.
namespace	Namespace to be associated with the database. This field is altered by changing the workspace name.
max connections	Maximum amount of open connections to the database.
min connections	Minimum number of pooled connections.
fetch size	Number of records read with each interaction with the database.
Connection timeout	Time (in seconds) the connection pool will wait before timing out.
validate connections	Checks the connection is alive before using it.

When finished, click *Save*.

Other data sources are supplied as GeoServer extensions. Extensions are downloadable modules that add functionality to GeoServer. Extensions are available at the GeoServer download page.

Warning: The extension version must match the version of the GeoServer instance.

5.2.5 GML

Note: GeoServer does not come built-in with support for GML; it must be installed through an extension. Proceed to *Installing the GML extension* for installation details.

Warning: Currently the GML extension is unmaintained and carries unsupported status. While still usable, do not expect the same reliability as with other extension.

Geographic Markup Language (GML) is a XML based format for representing vector based spatial data.

Supported versions

Currently GML version 2 is supported.

Installing the GML extension

1. Download the GML extension from the GeoServer download page.

   Warning: Make sure to match the version of the extension to the version of the GeoServer instance!

2. Extract the contents of the archive into the `WEB-INF/lib` directory of the GeoServer installation.

Adding a GML data store

Once the extension is properly installed *GML* will be an option in the *Vector Data Sources* list when creating a new data store.

Fig. 5.57: *GML in the list of vector data stores*

Configuring a GML data store

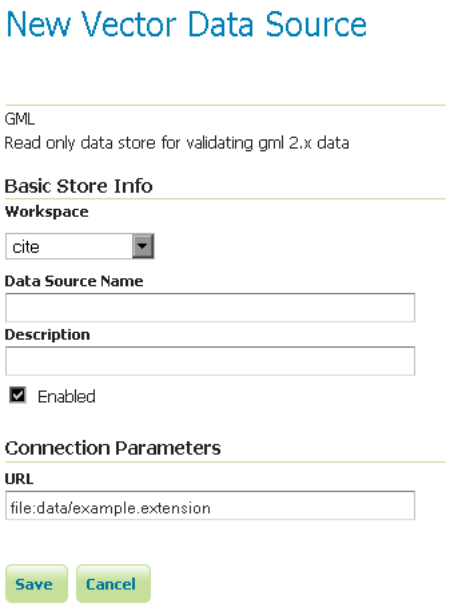


Fig. 5.58: *Configuring a GML data store*

5.2.6 Pregeneralized Features

Note: GeoServer does not come built-in with support for Pregeneralized Features; it must be installed through an extension.

Installing the Pregeneralized Features extension

1. Download the Pregeneralized Features extension from the GeoServer download page.

> **Warning:** Make sure to match the version of the extension to the version of the GeoServer instance!

2. Extract the contents of the archive into the `WEB-INF/lib` directory of the GeoServer installation.

Adding a Pregeneralized Features data store

If the extension is properly installed, *Generalized Data Store* will be listed as an option when creating a new data store.

Fig. 5.59: *Generalized Data Store in the list of vector data stores*

Configuring a Pregeneralized Features data store

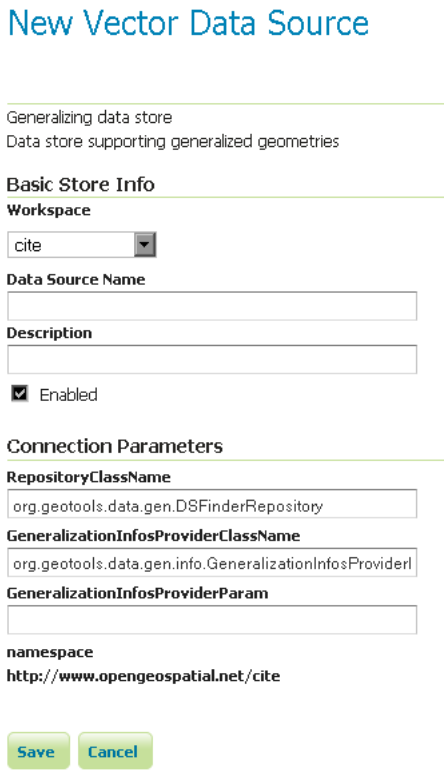


Fig. 5.60: *Configuring a Pregeneralized Features data store*

For a detailed description, look at the *Tutorial*

5.3 Raster data

This section discusses the raster (coverage) data sources that GeoServer can access.

The standard GeoServer installation supports the loading and serving of the following data formats:

5.3.1 GeoTIFF

A GeoTIFF is a georeferenced TIFF (Tagged Image File Format) file.

Adding a GeoTIFF data store

By default, *GeoTIFF* will be an option in the *Raster Data Sources* list when creating a new data store.

Fig. 5.61: *GeoTIFF in the list of raster data stores*

Configuring a GeoTIFF data store

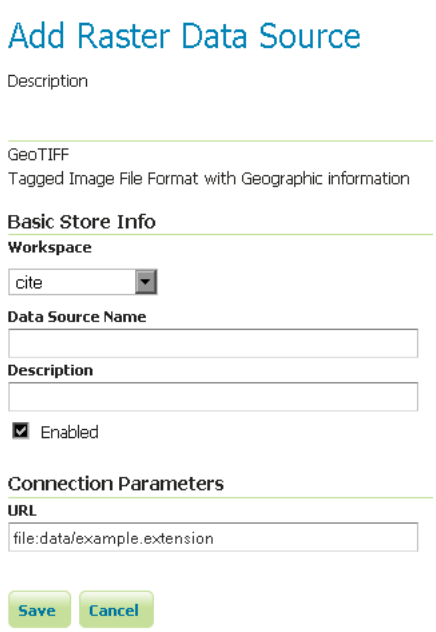


Fig. 5.62: *Configuring a GeoTIFF data store*

Option	Description
`Workspace`	Name of the workspace to contain the GeoTIFF store. This will also be the prefix of the raster layer created from the store.
`Data Source Name`	Name of the GeoTIFF as it will be known to GeoServer. This can be different from the filename. The combination of the workspace name and this name will be the full layer name (ex: world:landbase)
`Description`	A full free-form description of the GeoTIFF store.
`Enabled`	If checked, it enables the store. If unchecked (disabled), no data in the GeoTIFF will be served from GeoServer.
`URL`	Location of the GeoTIFF file. This can be an absolute path (such as `file:C:\Data\landbase.tif`) or a path relative to GeoServer's data directory (such as `file:data/landbase.tif`).

Note: Notice that the GeoTiff plugin is able to handle internal/external overviews and internal/external masks.

Custom CRS definition

Creating an auxiliary `.prj` file that contains coordinate reference system information as described in the *Custom CRS Definitions* chapter will override internal CRS tags that are included in the original GeoTIFF file. This can be used to work-around problematic source files without making modifications to the file.

5.3.2 GTOPO30

GTOPO30 is a Digital Elevation Model (DEM) dataset with a horizontal grid spacing of 30 arc seconds.

Note: An example of a GTOPO30 can be found at http://edc.usgs.gov/products/elevation/gtopo30/gtopo30.html

Adding a GTOPO30 data store

By default, *GTOPO30* will be an option in the *Raster Data Sources* list when creating a new data store.

Fig. 5.63: *GTOPO30 in the list of raster data stores*

Configuring a GTOPO30 data store

Option	Description
`Workspace`	
`Data Source Name`	
`Description`	
`Enabled`	
`URL`	

5.3.3 WorldImage

A world file is a plain text file used to georeference raster map images. This file (often with an extension of `.jgw` or `.tfw`) accompanies an associated image file (`.jpg` or `.tif`). Together, the world file and the corresponding image file is known as a WorldImage in GeoServer.

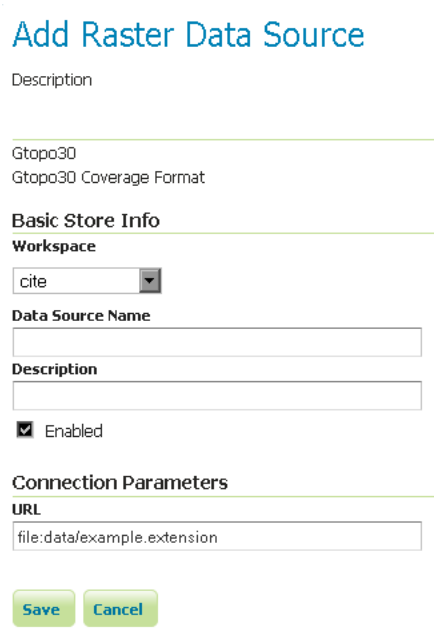


Fig. 5.64: *Configuring a GTOPO30 data store*

Adding a WorldImage data store

By default, *WorldImage* will be an option in the *Raster Data Sources* list when creating a new data store.

Fig. 5.65: *WorldImage in the list of raster data stores*

Configuring a WorldImage data store

Option	Description
`Workspace`	
`Data Source Name`	
`Description`	
`Enabled`	
`URL`	

5.3.4 ImageMosaic

The ImageMosaic data store allows the creation of a mosaic from a number of georeferenced rasters.

The mosaic operation creates a mosaic from two or more source images. This operation could be used to assemble a set of overlapping geospatially rectified images into a contiguous image. It could also be used to create a montage of photographs such as a panorama.

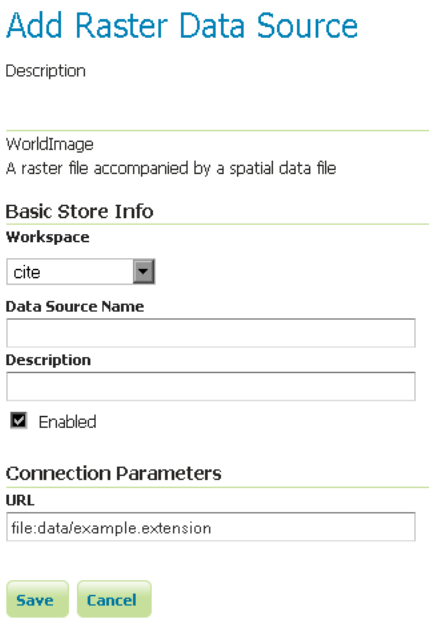


Fig. 5.66: *Configuring a WorldImage data store*

The plugin can be used with GeoTIFFs, as well as rasters accompanied by a world file (`.wld`, `.pgw` for PNG files, `.jgw` for JPG files, etc.). In addition, if imageIO-ext *GDAL extension* is properly installed, the plugin can also serve all the formats supported by it such as MrSID, ECW, JPEG2000. It also supports NetCDF and GRIB.

ImageMosaic configuration

Granules

Each individual image is commonly referred to as a **granule**. In recent releases of GeoServer the similarities requirements for the granules have been dropped significantly, including:

- The granules do not need to share the same coordinate reference system (see *the multi-CRS mosaic tutorial*)
- The granules can be in different color models, with an allowance of mixing gray, RGB, RGBA and indexed color granules (it is however not possible to mix colored granules with scientific data types like as float/double). In order to benefit of mixed color models JAI-Ext support must be enabled, see *the JAI-EXT support documentation*.

In addition it is worth remarking on the fact that currently the ImageMosaic is able to handle raster data whose grid-to-world transformation is a scale and translate transformation, hence no rotation or skew.

Index and configuration file creation

When a new store is created, an index shapefile will be generated to associate each granule file with its bounding box. The index creation respects directory trees as well as single directories. All you need to

do is point the store to the root of the hierarchy, and all images will be considered for inclusion in the ImageMosaic.

The index will contain the enclosing polygon for each raster file (in an appropriate coordinate reference system) and the path to each of these files. The location attribute can be relative to the configuration folder or absolute. By default, the name of this attribute is `location`, but this can be changed in the main configuration file.

If you already have these files generated, GeoServer will respect them and not generate a new index. By default, a shapefile is used for the index, but PostGIS, H2, and Oracle are also supported, with additional configuration steps.

Configuration files

Within each store there are multiple configuration files that determine how the mosaic is rendered.

Primary configuration file

The mosaic configuration file is the primary file used to store the configuration parameters that control the ImageMosaic plugin. When created by GeoServer it is by default called `<directory>.properties`, where `<directory>` is the name of the root directory of the store. (It is not related to the store name in GeoServer.) It can have other names, as long as it does not conflict with other files such as `datastore.properties` or `indexer.properties`. This file usually does not require manual editing.

The table below describes the various elements in this configuration file.

Parameter	Mandatory?	Description
Levels	Y	Represents the resolutions for the various levels of the granules of this mosaic.
Heterogeneous	N	Sets whether the image files are heterogeneous. Default is `false`.
AbsolutePath	N	Controls whether or not the path stored inside the `location` attribute represents an absolute path or a path relative to the location of the shapefile index. Notice that a relative index ensures much more portability of the mosaic itself. Default value for this parameter is `false`, which means relative paths.
Name	N	The name to be assigned to the index. If unspecified, the index name will usually match the name of the folder containing the mosaic.
TypeName	Y	Featuretype name for this mosaic. Usually the name as `Name`.
Caching	N	Boolean value to enable caching. When set to `true`, the ImageMosaic will try to save in memory the entire contents of the index to reduce loading/query time. Set to `false` for a large granule index and/or if new granules are to be ingested (for example, when the index is on a database and we interact directly with it). Default is `false`.
ExpandToRGB	N	Boolean flag to force color expansion from index color model (paletted datasets) to component color model (RGB). Default is `false`.
LocationAttribute	Y	The name of the attribute path in the shapefile index. Default is `location`.
SuggestedSPI	Y	Suggested plugin for reading the image files.
Envelope2D	N	Envelope for the mosaic formatted as `LLX,LLY URX,URY` (notice the space between the lower left and upper right coordinate pairs).
CheckAuxiliaryMetadata	N	This parameter allows to specify whether the ImageMosaic plugin should check for the presence of a GDAL aux.xml file beside each granule file. For most common use cases, you don't need to set or specify this parameter. Being disabled by Default, ImageMosaic won't look for an ancillary file for each granule being initialized in the GranuleCatalog. This avoid useless checks, especially when dealing with thousand of granules. You should set that parameter to `true` when you want to instruct the ImageMosaic to look for a GDAL generated aux.xml file containing PAM (Persistent Auxiliary Metadata) for each granule, to be attached to the Granule info (GranuleDescriptor). This is specially useful when you have setup a *Dynamic ColorMap rendering transformation* which dynamically set a color map based on the statistics collected into the granule's GDAL PAM being previously generated with a gdalinfo -stats parameter.
LevelsNum	Y	Represents the number of reduced resolution layers that we currently have for the granules of this mosaic.

A sample configuration file follows:

```
Levels=0.4,0.4
Heterogeneous=false
AbsolutePath=false
Name=osm
TypeName=osm
Caching=false
ExpandToRGB=false
LocationAttribute=location
SuggestedSPI=it.geosolutions.imageioimpl.plugins.tiff.TIFFImageReaderSpi
CheckAuxiliaryMetadata=false
LevelsNum=1
```

datastore.properties

By default the ImageMosaic index is specified by a shapefile, which is located at the root of the ImageMosaic directory, just like the primary configuration file.

If needed, different storage can be used for the index — like a spatial DBMS, which is the preferred solution when you wish to share the ImageMosaic itself in a cluster of GeoServer instances. In this case the user must supply GeoServer with the proper connection parameters, which can be specified by using a `datastore.properties` file placed at the root of the ImageMosaic directory.

Note: A shapefile is created automagically if it does not exist or if there is no `datastore.properties` file.

Warning: At the time of writing the following spatial DBMS have been tested successfully: Oracle, PostgreSQL, H2. SQl Server is not yet supported.

Parameter	Mandatory?	Description
StoreName	N	Can be used to refer to a GeoServer registered store, using a "workspace:storeName" syntax. When this is used, the no other connection parameters need to be provided. The SPI can still be provided to inform the mosaic of the resulting type of store (e.g., Oracle) in case specific behavior need to be enacted for it (e.g., in the case of Oracle the attributes are all uppercase and cannot be longer than 30 chars, the mosaic will respect the limits but the *SPI* parameter needs to be explicitly set to *org.geotools.data.oracle.OracleNGDataStoreFactory* as the actual store type is hidden when it reaches the mosaic code). Also, as a reminder, the code is picking up a Store reference, not a layer one, meaning that security restrictions that might have been applied to a layer exposing the feature type do not apply to the mosaic code (e.g., if a user has restrictions such as a spatial filter on said layer, it won't transfer to the mosaic, which needs to be secured separately)
SPI	Y	The DataStoreFactory used to connect to the index store: • PostGIS: `org.geotools.data.postgis.PostgisNGDataStoreFactory` • Oracle: `org.geotools.data.oracle.OracleNGDataStoreFactory` • H2: `org.geotools.data.h2.H2DataStoreFactory` *JNDI* can also be used with any of these stores. If JNDI is used, the DataStoreFactory name will differ from the above.
Connection parameters	Y	The connection parameters used by the specified SPI. The list of these connection parameters can be found in the GeoTools documentation on the relevant store: • PostGIS • Oracle • H2 If JNDI is used, the connection parameters will include `jndiReferenceName` instead of `host`, `port`, etc. Note that for any connection parameters that include a space (such as `loose bbox`), the space must be escaped by preceding it with a backslash (`loose\ bbox`).

Here is a sample `datastore.properties` file for a PostGIS index:

```
SPI=org.geotools.data.postgis.PostgisNGDataStoreFactory
host=localhost
port=5432
```

```
database=osm
schema=public
user=user
passwd=password
Loose\ bbox=true
Estimated\ extends=false
validate\ connections=true
Connection\ timeout=10
preparedStatements=true
```

Here is a sample `datastore.properties` file for a PostGIS index via JNDI:

```
SPI=org.geotools.data.postgis.PostgisNGJNDIDataStoreFactory
#String
# JNDI data source
# Default "java:comp/env/"+"jdbc/mydatabase"
jndiReferenceName=

#Boolean
# perform only primary filter on bbox
# Default Boolean.TRUE
Loose\ bbox=true

#Boolean
# use prepared statements
#Default Boolean.FALSE
preparedStatements=false
```

indexer.properties

In addition to the required envelope and location attributes, the schema for the index store may expose other custom attributes which can be used later for filtering the ImageMosaic granules on the fly during a WMS or WCS request or to diver WMS and WCS dimensions like TIME, ELEVATION and so on. This is configured by the `indexer.properties` file:

Parameter	Mandatory?	Description
Schema	Y	A comma-separated sequence describing the mapping between attribute and data type.
PropertyCollectors	Y	A comma-separated list of PropertyCollectors. Each entry in the list includes the extractor class, the file name (within square brackets `[ ]` and not including the `.properties` suffix) containing the regular expression needed to extract the attribute value from the granule file name, and the attribute name (within parentheses `( )`). The instance of the extractor class also indicates the type of object computed by the specific collector, so a `TimestampFileNameExtractorSPI` will return `Timestamps` while a `DoubleFileNameExtractorSPI` will return `Double` numbers.
TimeAttribute	N	Specifies the name of the time-variant attribute.
ElevationAttribute	N	Specifies the name of the elevation attribute.
AuxiliaryFile	N	Path to an auxiliary file to be used for internal purposes (For example: when dealing with NetCDF granules, it refers to the NetCDF XML ancillary file.)
AbsolutePath	N	Controls whether or not the path stored inside the `location` attribute represents an absolute path or a path relative to the location of the shapefile index. Notice that a relative index ensures better portability of the mosaic itself. Default value for this parameter is `false`, which means relative paths.
Caching	N	Boolean value to enable caching. When set to `true`, the ImageMosaic will try to save in memory the entire contents of the index to reduce loading/query time. Set to `false` for a large granule index and/or if new granules are to be ingested (for example, when the index is on a database and we interact directly with it). Default is `false`.
CanBeEmpty	N	Boolean flag used for configuring empty mosaics. When enabled the ImageMosaic will not throw an exception caused by the absence of any coverage. By default it is set to `false`.
Envelope2D	N	Envelope for the mosaic formatted as `LLX,LLY URX,URY` (notice the space between the lower left and upper right coordinate pairs).
ExpandToRGB	N	Boolean flag to force color expansion from index color model (paletted datasets) to component color model (RGB). Default is `false`.
IndexingDirectories	N	Comma separated values list of paths referring to directories containing granules to be indexed. If unspecified, the IndexingDirectory will be the mosaic configuration directory. This parameter allows configuration of a mosaic in a folder which contains only configuration files, while the granules to be indexed are stored somewhere else.
Name	N	The name to be assigned to the index. If unspecified, the index name will usually match the name of the folder containing the mosaic.
CoverageNameCollectorSPI	N	As described in the previous row, the Name parameter allows specification of the coverage name to be exposed by the ImageMosaic. An ImageMosaic of NetCDFs instead exposes a coverage for each supported variable found in the NetCDF, using the variable's name as the coverage name (for instance, air_temperature, wind_speed, etc.) The optional CoverageNameCollectorSPI property allows specification of a CoverageNameCollector plugin to be used to instruct the ImageMosaic on how to setup different coverageNames for granules. It should contains the full name of the implementing class plus an optional set of semicolon-separated key-Value pairs prefixed by ":". See below for an example.
Recursive	N	Boolean flag used at indexing time. When set to `true`, the indexer will look for granules by scanning any subdirectory contained in the indexing directory. If `false`, only the main folder will be analyzed. Default is `true`.
UseExistingSchema	N	Boolean flag used for enabling/disabling the use of existing schemas. When enabled, the ImageMosaic will start indexing granules using the existing database schema (from `datastore.properties`) instead of populating it. This is useful when you already have a database with a valid mosaic schema (the_geom, location and other attributes, take a look at gdalindex) or when you do not want to rename the images to add times and dimensions (you should simply add them to the table, to AdditionalDomainAttributes and to PropertyCollectors). Default is `false`.
Wildcard	N	Wildcard used to specify which files should be scanned by the indexer. (For instance: ".")

Here is a sample `indexer.properties` file:

```
Schema=*the_geom:Polygon,location:String,ingestion:java.util.Date,elevation:Double
PropertyCollectors=TimestampFileNameExtractorSPI[timeregex](ingestion),
↪DoubleFileNameExtractorSPI[elevationregex](elevation)
TimeAttribute=ingestion
ElevationAttribute=elevation
Caching=false
AbsolutePath=false
```

An example of optional CoverageNameCollectorSPI could be:

```
CoverageNameCollectorSPI=org.geotools.gce.imagemosaic.namecollector.
↪FileNameRegexNameCollectorSPI:regex=^([a-zA-Z0-9]+)
```

This defines a regex-based name collector which extracts the coverage name from the prefix of the file name, so that an ImageMosaic with temperature_2015.tif, temperature_2016.tif, pressure_2015.tif, pressure_2016.tif will put temperature* granules on a `temperature` coverage and pressure* granules on a `pressure` coverage.

Property collectors

The following table enumerates the available property collectors

Collector SPI name	Description
ByteFileNameExtractorSPI DoubleFileNameExtractorSPI FloatFileNameExtractorSPI IntegerFileNameExtractorSPI LongFileNameExtractorSPI ShortFileNameExtractorSPI	Extracts an number from the file name using a regular expression specified in a sidecar file, casting it to the desired type based on the SPI name (e..g, DoubleFileNameExtractorSPI extracts double precision floating points, IntegerFileNameExtractorSPI extracts 32 bit integers)
TimestampFileNameExtractorSPI	Extracts a timestamp from the filename using a regular expression specified in a sidecar file
StringFileNameExtractorSPI	Extracts a string from the filename using a regular expression specified in a sidecar file
CurrentDateExtractorSPI	Returns the current date and time (useful to track ingestion times in a mosaic)
FSDateExtractorSPI	Returns the creation date of the file being harvested
DateExtractorSPI	Returns the date found in tiff file header "DateTime" (code 306)
ResolutionExtractorSPI ResolutionXExtractorSPI ResolutionYExtractorSPI	Returns the native resolution of the raster being harvested. ResolutionExtractorSPI and ResolutionXExtractorSPI return the x resolution of the raster, ResolutionYExtractorSPI returns the resolution on the Y axis instead
CRSExtractorSPI	Returns the code of the the raster coordinate reference system, as a string, e.g. "EPSG:4326"

The `PropertyCollectors` parameter in the example above indicates two additional `.properties` files used to populate the `ingestion` and `elevation` attributes:

`timeregex.properties`:

```
regex=[0-9]{8}T[0-9]{9}Z(\?!.\*[0-9]{8}T[0-9]{9}Z.\*)
```

The above is a property file containing a regex used to extract Date and Time represented in ISO-8601 as part of the filename. (Note the T char between digits for date and digits for time, as per ISO-8601)

In case of custom format datetimes in filename, an additional *format* element should be added after the regex, preceded by a comma, defining the custom representation.

Example:
Temperature_2017111319.tif
an hourly Temperature file with datetime = November, 13 2017 at 7:00 PM (the last 2 digits = 19)

In that case, the timeregex.properties file should be like this:

regex=.*([0-9]{10}).*,format=yyyyMMddHH

`elevationregex.properties`:

```
regex=(?<=_)(\\d{4}\\.\\d{3})(?=_)
```

Store parameters

By default, *ImageMosaic* will be an option in the *Raster Data Sources* list when creating a new data store.

Fig. 5.67: ImageMosaic in the list of raster data stores

Option	Description
Workspace	Workspace for the store
Data Source Name	Name of the store
Description	Description of the store
Enabled	Determines whether the store is enabled. If unchecked, all layers in the store will be disabled.
URL	The location of the store. Can be a local directory.

Coverage parameters

Creation of the store is the first step to getting an ImageMosaic published in GeoServer. Most of the configuration is done when publishing the resulting coverage (layer).

The Coverage Editor gives users the possibility to set a few control parameters to further control the mosaic creation process.

The parameters are as follows:

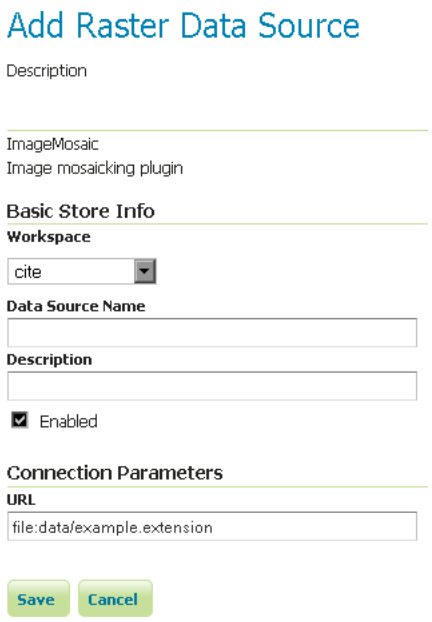


Fig. 5.68: Configuring an ImageMosaic data store

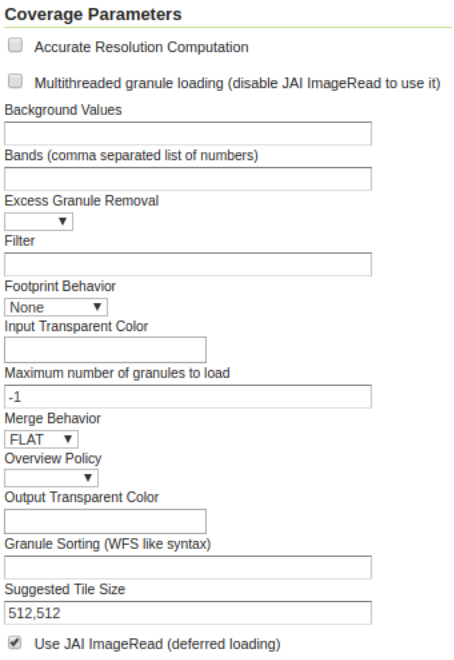


Fig. 5.69: Coverage parameters

Parameter	Description
Accurate resolution computation	Boolean value. If `true`, computes the resolution of the granules in 9 points: the corners of the requested area and the middle points, taking the better one. This will provide better results for cases where there is a lot more deformation on a subregion (top/bottom/sides) of the requested bounding box with respect to others. If `false`, computes the resolution using a basic affine scale transform.
AllowMultithreading	If `true`, enables multithreaded tile loading. This allows performing parallelized loading of the granules that compose the mosaic. Setting this to `true` makes sense only if you set USE_JAI_IMAGEREAD to `false` at the same time to force immediate loading of data into memory.
BackgroundValues	Sets the value of the mosaic background. Depending on the nature of the mosaic it is wise to set a value for the "nodata" area (usually -9999). This value is repeated on all the mosaic bands.
Filter	Sets the default mosaic filter. It should be a valid *ECQL query* to be used by default if no `cql_filter` is specified (instead of Filter.INCLUDE). This filter will be applied against the mosaic index, and may include any attributes exposed by the index store. If the `cql_filter` is specified in the request it will be overridden. **Note:** Do not use this filter to change time or elevation dimensions defaults. It will be added as AND condition with CURRENT for "time" and LOWER for "elevation".
FootprintBehavior	Sets the behavior of the regions of a granule that are outside of the granule footprint. Can be `None` (ignore the footprint), `Cut` (remove regions outside the footprint from the image and don't add an alpha channel), or `Transparent` (make regions outside the footprint completely transparent, and add an alpha channel if one is not already present). Defaults to `None`.
InputTransparentColor	Sets the transparent color of the granules prior to processing by the ImageMosaic plugin, in order to control how they are superimposed. When GeoServer composes the granules to satisfy a user request, some can overlap others; setting this parameter with an appropriate color avoids the overlap of "nodata" areas between granules. See below for an example:  Fig. 5.70: InputTransparentColor parameter not configured
MaxAllowedTiles	Sets the maximum number of tiles that can be loaded simultaneously for a request. For large mosaics, this parameter should be set to avoid saturating the server by loading too many granules simultaneously.
MergeBehavior	The method used to handle overlapping granules during the mosaic operation. Can be `FLAT` (only the topmost granule is visible in the case of an overlap) or `STACK` (a band-stacking merge is applied to the overlapping granules). Default is `FLAT`.
OutputTransparentColor	Set the transparent color for the mosaic. This parameter make sense for RGB or paletted mosaics, but not for a DEM or MetOc data. See below for an example: 

Continue on with the *ImageMosaic tutorial* to learn more and see examples.

Using the ImageMosaic extension

This tutorial will show you how to configure and publish an ImageMosaic store and coverage, followed by some configuration examples.

Configuring a coverage in GeoServer

This is a process very similar to creating a featuretype. More specifically, one has to perform the steps highlighted in the sections below:

Create a new store

1. Go to *Data Panel* → *Stores* and click *Add new Store*.
2. Select *ImageMosaic* under *Raster Data Source*:

Fig. 5.74: ImageMosaic in the list of raster data stores

3. In order to create a new mosaic it is necessary to choose a workspace and store name in the *Basic Store Info* section, as well as a URL in the *Connection Parameters* section. Valid URLs include:
   - The absolute path to the shapefile index, or a directory containing the shapefile index.
   - The absolute path to the configuration file (**.properties'*) or a directory containing the configuration file. If `datastore.properties` and `indexer.properties` exist, they should be in the same directory as this configuration file.
   - The absolute path of a directory where the files you want to mosaic reside. In this case GeoServer automatically creates the needed mosaic files (.dbf, .prj, .properties, .shp and .shx) by inspecting the data present in the given directory and any subdirectories.
4. Click *Save*:

Create a new coverage

1. Navigate to *Data Panel* → *Layers* and click *Add a new resource*.
2. Choose the name of the store you just created:
3. Click the layer you wish to configure and you will be presented with the Coverage Editor:
4. Make sure there is a value for *Native SRS*, then click the *Submit* button. If the *Native CRS* is `UNKNOWN`, you must declare the SRS in the *Declared SRS* field.
5. Click *Save*.
6. Use the *Layer Preview* to view the mosaic.

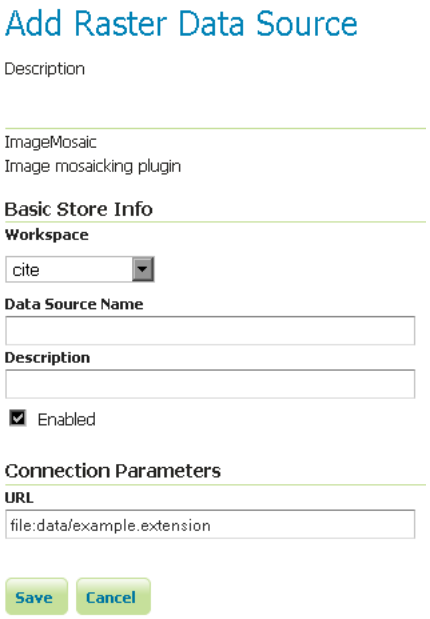


Fig. 5.75: Configuring an ImageMosaic data store

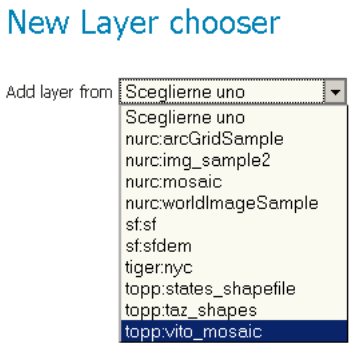


Fig. 5.76: Layer Chooser

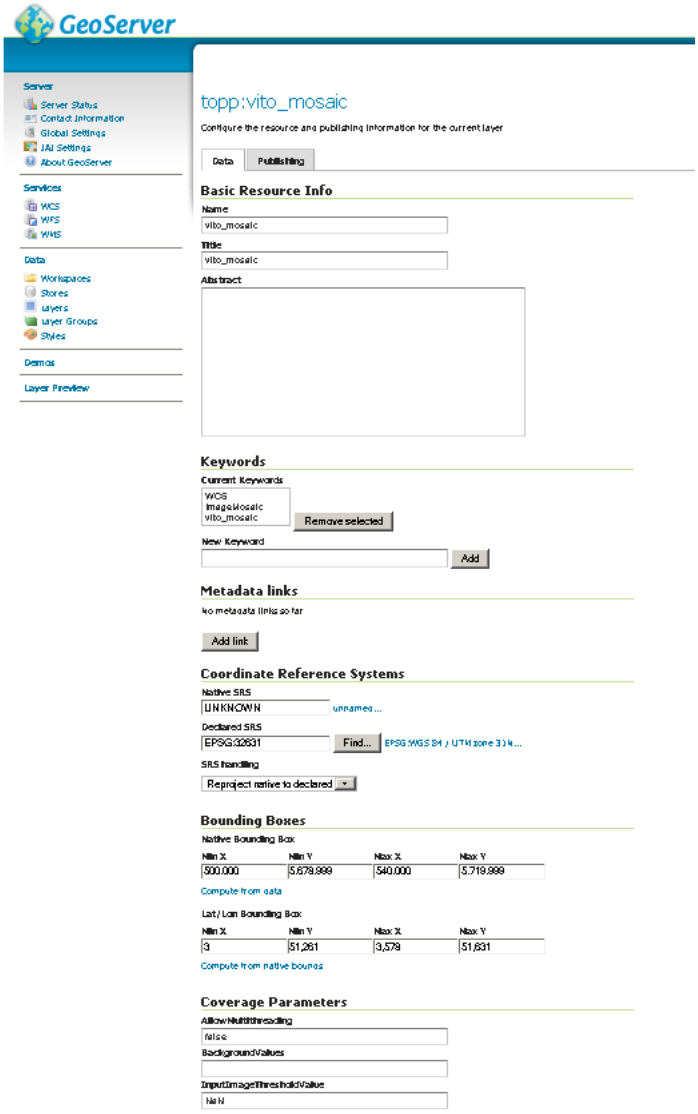


Fig. 5.77: Coverage Editor

Warning: If the created layer appears to be all black, it may be that GeoServer has not found any acceptable granules in the provided index. It is also possible that the shapefile index is empty (no granules were found in the provided directory) or it might be that the granules' paths in the shapefile index are not correct, which could happen if an existing index (using absolute paths) is moved to another place. If the shapefile index paths are not correct, then the DBF file can be opened and fixed with an editor. Alternately, you can delete the index and let GeoServer recreate it from the root directory.

Configuration examples

Below are a few examples of mosaic configurations to demonstrate how we can make use of the ImageMosaic parameters.

DEM/Bathymetry

Such a mosaic can be used to serve large amounts of data representing altitude or depth and therefore does not specify colors directly (it needs an SLD to generate pictures). In our case, we have a DEM dataset which consists of a set of raw GeoTIFF files.

The first operation is to create the CoverageStore specifying, for example, the path of the shapefile in the *URL* field.

Inside the Coverage Editor Publishing tab, you can specify the *dem* default style in order to represent the visualization style of the mosaic. The following is an example style:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<StyledLayerDescriptor version="1.0.0"
  xmlns="http://www.opengis.net/sld" xmlns:ogc="http://www.opengis.net/ogc"
  xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xsi="http://www.w3.org/2001/
↪XMLSchema-instance"
  xsi:schemaLocation="http://www.opengis.net/sld      http://schemas.opengis.net/sld/
↪1.0.0/StyledLayerDescriptor.xsd">
  <NamedLayer>
    <Name>gtopo</Name>
    <UserStyle>
      <Name>dem</Name>
      <Title>Simple DEM style</Title>
      <Abstract>Classic elevation color progression</Abstract>
      <FeatureTypeStyle>
        <Rule>
          <RasterSymbolizer>
            <Opacity>1.0</Opacity>
            <ColorMap>
              <ColorMapEntry color="#000000" quantity="-9999" label="nodata" opacity=
↪"1.0" />
              <ColorMapEntry color="#AAFFAA" quantity="0" label="values" />
              <ColorMapEntry color="#00FF00" quantity="1000" label="values" />
              <ColorMapEntry color="#FFFF00" quantity="1200" label="values" />
              <ColorMapEntry color="#FF7F00" quantity="1400" label="values" />
              <ColorMapEntry color="#BF7F3F" quantity="1600" label="values" />
              <ColorMapEntry color="#000000" quantity="2000" label="values" />
            </ColorMap>
          </RasterSymbolizer>
        </Rule>
      </FeatureTypeStyle>
```

```
    </UserStyle>
  </NamedLayer>
</StyledLayerDescriptor>
```

In this way you have a clear distinction between the different intervals of the dataset that compose the mosaic, like the background and the "nodata" area.

Note: The "nodata" on the sample mosaic is -9999. The default background value is for mosaics is 0.0.

The result is the following:

By setting the other configuration parameters appropriately, it is possible to improve both the appearance of the mosaic as well as its performance. For instance, we could:

- Make the "nodata" areas transparent and coherent with the real data. To achieve this we need to change the opacity of the "nodata" ColorMapEntry in the `dem` style to `0.0` and set the `BackgroundValues` parameter to `-9999` so that empty areas will be filled with this value. The result is as follows:
- Allow multithreaded granules loading. By setting the `AllowMultiThreading` parameter to `true`, GeoServer will load the granules in parallel using multiple threads with a increase in performance on some architectures.

The configuration parameters are as follows:

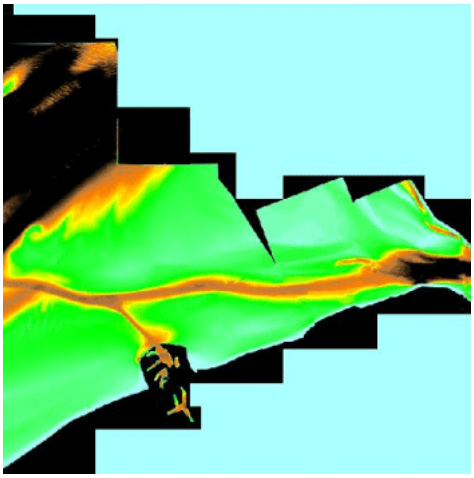


Fig. 5.78: Basic configuration

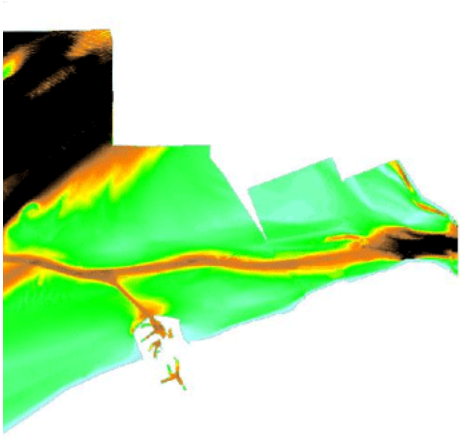


Fig. 5.79: Advanced configuration

Parameter	Value
MaxAllowedTiles	2147483647
BackgroundValues	-9999
OutputTransparentColor	"no color"
InputTransparentColor	"no color"
AllowMultiThreading	True
USE_JAI_IMAGEREAD	True
SUGGESTED_TILE_SIZE	512,512

Aerial imagery

In this example we are going to create a mosaic that will serve aerial imagery, specifically RGB GeoTIFFs. Because this is visual data, in the Coverage Editor you can use the basic `raster` style, which is just a stub SLD to instruct the GeoServer raster renderer to not do anything particular in terms of color management:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<StyledLayerDescriptor version="1.0.0"
  xmlns="http://www.opengis.net/sld" xmlns:ogc="http://www.opengis.net/ogc"
  xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xsi="http://www.w3.org/2001/
↪XMLSchema-instance"
  xsi:schemaLocation="http://www.opengis.net/sld      http://schemas.opengis.net/sld/
↪1.0.0/StyledLayerDescriptor.xsd">
  <NamedLayer>
    <Name>raster</Name>
    <UserStyle>
      <Name>raster</Name>
      <Title>Raster</Title>
      <Abstract>A sample style for rasters, good for displaying imagery       </
↪Abstract>
      <FeatureTypeStyle>
        <FeatureTypeName>Feature</FeatureTypeName>
        <Rule>
          <RasterSymbolizer>
            <Opacity>1.0</Opacity>
          </RasterSymbolizer>
        </Rule>
      </FeatureTypeStyle>
    </UserStyle>
  </NamedLayer>
</StyledLayerDescriptor>
```

The result is the following:

Note: Those ugly black areas are the result of applying the default mosaic parameters to a mosaic that does not entirely cover its bounding box. The areas within the BBOX that are not covered with data will default to a value of 0 on each band. Since this mosaic is RGB we can simply set the `OutputTransparentColor` to `0,0,0` in order to get transparent fills for the BBOX.

The various parameters can be set as follows:

Fig. 5.80: Basic configuration

Parameter	Value
MaxAllowedTiles	2147483647
BackgroundValues	(default)
OutputTransparentColor	#000000
InputTransparentColor	"no color"
AllowMultiThreading	True
USE_JAI_IMAGEREAD	True
SUGGESTED_TILE_SIZE	512,512

The result is the following:

Fig. 5.81: Advanced configuration

Scanned maps

In this case we want to show how to serve scanned maps (mostly B&W images) via a GeoServer mosaic.

In the Coverage Editor you can use the basic `raster` since there is no need to use any of the advanced RasterSymbolizer capabilities.

The result is the following.

Fig. 5.82: Basic configuration

This mosaic, formed by two single granules, shows a typical case where the "nodata" collar areas of the granules overlap, as shown in the picture above. In this case we can use the `InputTransparentColor` parameter to make the collar areas disappear during the superimposition process — in this case, by using an `InputTransparentColor` of `#FFFFFF`.

The final configuration parameters are the following:

Parameter	Value
MaxAllowedTiles	2147483647
BackgroundValues	(default)
OutputTransparentColor	"no color"
InputTransparentColor	#FFFFFF
AllowMultiThreading	True
USE_JAI_IMAGEREAD	True
SUGGESTED_TILE_SIZE	512,512

This is the result:

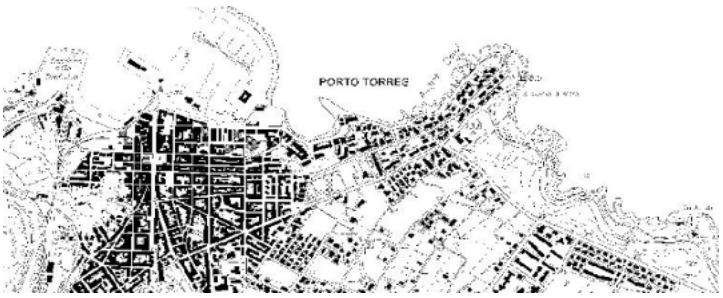


Fig. 5.83: Advanced configuration

Dynamic imagery

A mosaic need not be static. It can contain granules which change, are added or deleted. In this example, we will create a mosaic that changes over time.

1. Create a mosaic in the standard way. (The specific configuration isn't important.)

Fig. 5.84: This mosaic contains 5 granules. Note that `InputTransparentColor` is set to `#FFFFFF` here.

To add new granules, the index that was created when the mosaic was originally created needs to be regenerated. There are two ways to do this:

- Manually through the file system
- Through the *REST* interface

To update an ImageMosaic through the file system:

1. Update the contents of the mosaic by copying the new files into place. (Subdirectories are acceptable.)
2. Delete the index files. These files are contained in the top level directory containing the mosaic files and include (but are not limited to) the following:
   - `<mosaic_name>.dbf`
   - `<mosaic_name>.fix`
   - `<mosaic_name>.prj`
   - `<mosaic_name>.properties`
   - `<mosaic_name>.shp`
   - `<mosaic_name>.shx`
3. *(Optional but recommended)* Edit the layer definition in GeoServer, making to sure to update the bounding box information (if changed).
4. Save the layer. The index will be recreated.

Note: Please see the REST section for information on *Uploading a new image mosaic*.

Multi-resolution imagery with reprojection

As a general rule, we want to have the highest resolution granules shown "on top", with the lower-resolution granules filling in the gaps as necessary.

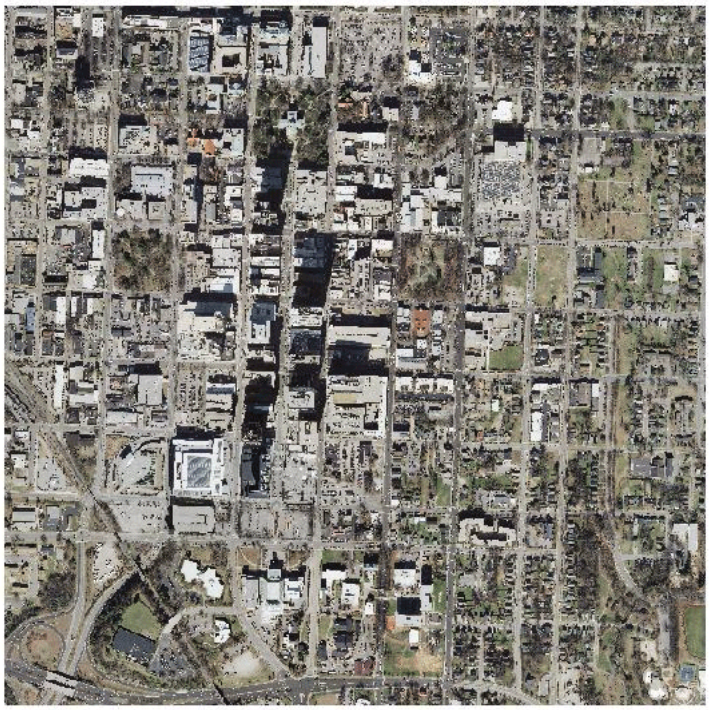


Fig. 5.85: This mosaic contains 9 granules

In this example, we will serve up overlapping granules that have varying resolutions. In addition, we will mix resolutions, such that the higher resolution granule is reprojected to match the resolution of the lower resolution granules.

1. In the Coverage Editor, use the basic `raster` style.
2. Create the mosaic in GeoServer.
3. One important configuration setting is the *SORTING* parameter of the layer. In order to see the highest resolution imagery on top (the typical case), it must be set to `resolution A`. (For the case of lowest resolution on top, use `resolution D`.)
4. Make any other configuration changes.
5. Also, in order to allow for multiple CRSs in a single mosaic, an `indexer.properties` file will need to be created. Use the following

```
GranuleAcceptors=org.geotools.gce.imagemosaic.acceptors.
↪HeterogeneousCRSAcceptorFactory
GranuleHandler=org.geotools.gce.imagemosaic.granulehandler.
↪ReprojectingGranuleHandlerFactory
HeterogeneousCRS=true
MosaicCRS=EPSG\:4326
PropertyCollectors=CRSExtractorSPI(crs),ResolutionExtractorSPI(resolution)
Schema=*the_geom:Polygon,location:String,crs:String,resolution:String
```

   The MosaicCRS property is not mandatory, but it's a good idea to set a predictable target CRS that all granule footprints can be reprojected into, otherwise the mosaic machinery will use the CRS of the first indexed granule.

6. Save this file in the root of the mosaic directory (along with the index files). The result is the following:

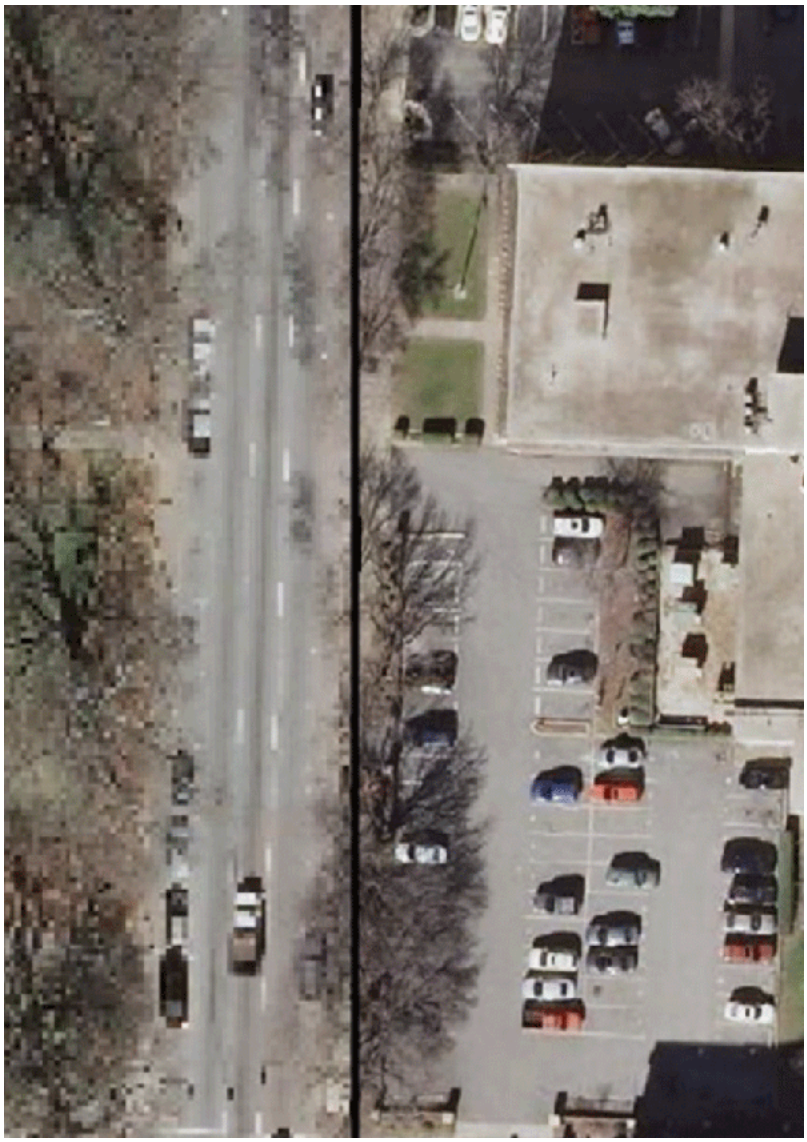


Fig. 5.86: Closeup of granule overlap (high resolution granule on right)

7. To remove the reprojection artifact (shown in the above as a black area) edit the layer configuration to set `InputTransparentColor` to `#000000`.

Referring to a datastore configured in GeoServer

It is possible to make the mosaic refer to an existing data store. The **"datastore.properties"** file in this case will contain only one or two properties, referring to the store to be used via the `StoreName` property. For simple cases, e.g., a PostGIS store, the following will be sufficient:

```
StoreName=workspace:storename
```

For Oracle or H2, it's best to also specify the SPI in order to inform the mosaic that it needs to work around specific limitations of the storage (e.g., forced uppercase attribute usage, limitation in attribute name length

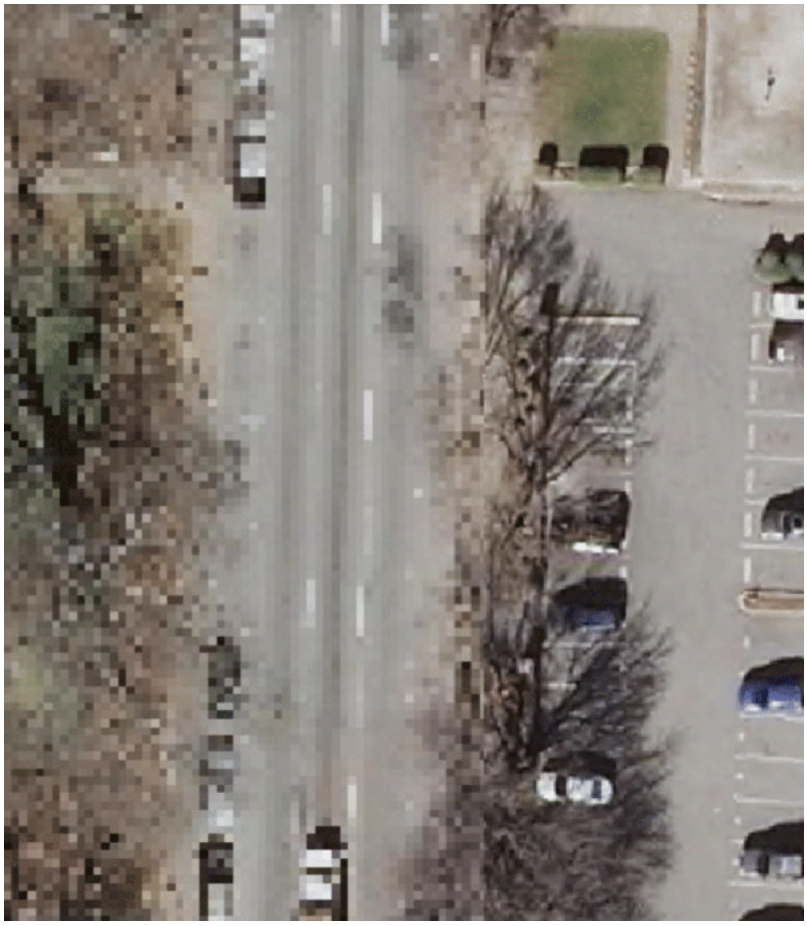


Fig. 5.87: Closeup of granule overlap (high resolution granule on right)

and the like):

```
StoreName=workspace:storename
SPI=org.geotools.data.oracle.OracleNGDataStoreFactory
```

The above will be sufficient in case the image mosaic can create the index table and perform normal indexing, using the directory name as the table name. In case a specific table name needs to be used, add an **"indexer.properties"** specifying the `TypeName` property, e.g.:

TypeName=myMosaicTypeName

In case the index "table" already exists instead, then a **"indexer.properties"** file will be required, with the following contents:

```
UseExistingSchema=true
TypeName=nameOfTheFeatureTypeContainingTheIndex
AbsolutePath=true
```

The above assumes `location` attribute provides absolute paths to the mosaic granules, instead of paths relative to the mosaic configuration files directory.

5.3.5 GeoPackage

GeoPackage is an SQLite based standard format that is able to hold multiple vector and raster data layers in a single file.

GeoPackage files can be used both as Vector Data Stores as well as Raster Data Stores (so that both kinds of layers can published).

Adding a GeoPackage Raster (Mosaic) Data Store

By default, *GeoPackage (mosaic)* will be an option in the *Raster Data Sources* list when creating a new data store.

Fig. 5.88: *GeoPackage (mosaic) in the list of raster data stores*

Option	Description
`Workspace`	Name of the workspace to contain the GeoPackage Mosaic store. This will also be the prefix of the raster layers created from the store.
`Data Source Name`	Name of the GeoPackage Mosaic Store as it will be known to GeoServer. This can be different from the filename.)
`Description`	A full free-form description of the GeoPackage Mosaic Store.
`Enabled`	If checked, it enables the store. If unchecked (disabled), no data in the GeoPackage Mosaic Store will be served from GeoServer.
`URL`	Location of the GeoPackage file. This can be an absolute path (such as `file:C:\Data\landbase.gpkg`) or a path relative to GeoServer's data directory (such as `file:data/landbase.gpkg`).

When finished, click *Save*.

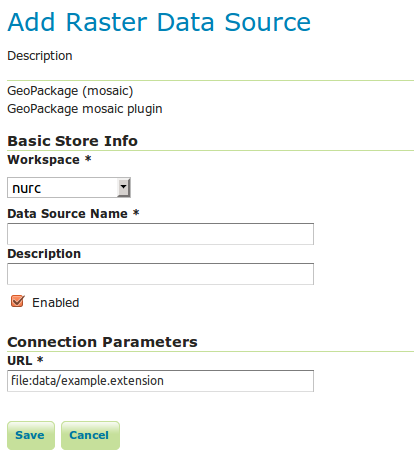


Fig. 5.89: *Configuring a GeoPackage (mosaic) data store*

Other data sources are supplied as GeoServer extensions. Extensions are downloadable modules that add functionality to GeoServer. Extensions are available at the GeoServer download page.

Warning: The extension version must match the version of the GeoServer instance.

5.3.6 ArcGrid

ArcGrid is a coverage file format created by ESRI.

Adding an ArcGrid data store

By default, *ArcGrid* will be an option in the *Raster Data Sources* list when creating a new data store.

Fig. 5.90: *ArcGrid in the list of raster data stores*

Configuring a ArcGrid data store

Option	Description
Workspace	
Data Source Name	
Description	
Enabled	
URL	

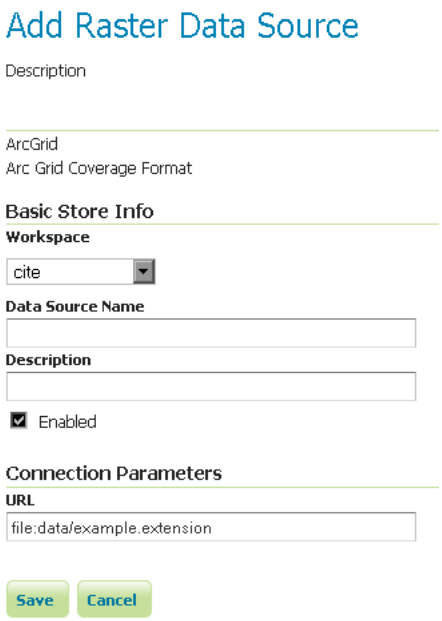


Fig. 5.91: *Configuring an ArcGrid data store*

5.3.7 GDAL Image Formats

GeoServer can leverage the ImageI/O-Ext GDAL libraries to read selected coverage formats. GDAL is able to read many formats, but for the moment GeoServer supports only a few general interest formats and those that can be legally redistributed and operated in an open source server.

The following image formats can be read by GeoServer using GDAL:

- DTED, Military Elevation Data (.dt0, .dt1, .dt2): http://www.gdal.org/frmt_dted.html
- EHdr, ESRI .hdr Labelled: <http://www.gdal.org/frmt_various.html#EHdr>
- ENVI, ENVI .hdr Labelled Raster: <http://www.gdal.org/frmt_various.html#ENVI>
- HFA, Erdas Imagine (.img): <http://www.gdal.org/frmt_hfa.html>
- JP2MrSID, JPEG2000 (.jp2, .j2k): <http://www.gdal.org/frmt_jp2mrsid.html>
- MrSID, Multi-resolution Seamless Image Database: <http://www.gdal.org/frmt_mrsid.html>
- NITF: <http://www.gdal.org/frmt_nitf.html>
- ECW, ERDAS Compressed Wavelets (.ecw): <http://www.gdal.org/frmt_ecw.html>
- JP2ECW, JPEG2000 (.jp2, .j2k): http://www.gdal.org/frmt_jp2ecw.html
- AIG, Arc/Info Binary Grid: <http://www.gdal.org/frmt_various.html#AIG>
- JP2KAK, JPEG2000 (.jp2, .j2k): <http://www.gdal.org/frmt_jp2kak.html>

Installing GDAL extension

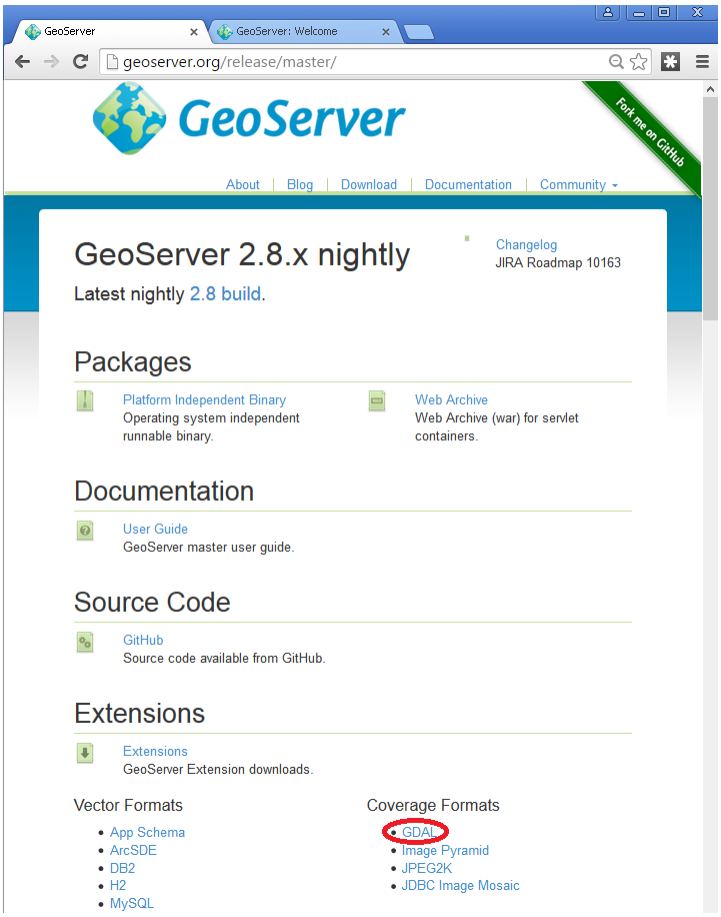
From GeoServer version 2.2.x, GDAL must be installed as an extension. To install it:

- Navigate to the GeoServer download page

- Find the page that matches the version of the running GeoServer.

> **Warning:** Be sure to match the version of the extension with that of GeoServer, otherwise errors will occur.

- Download the GDAL extension. The download link for *GDAL* will be in the *Extensions* section under *Coverage Format*.



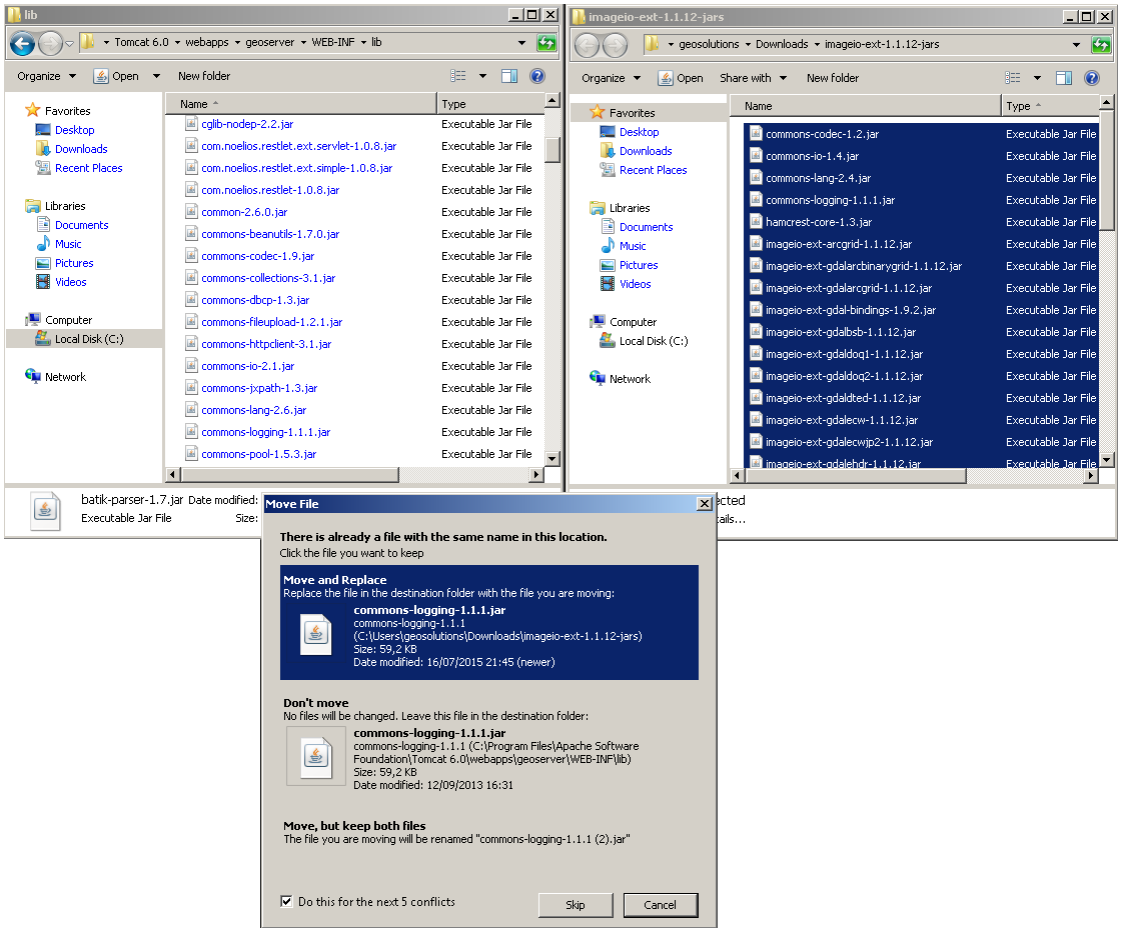
- Extract the files in this archive to the `WEB-INF/lib` directory of your GeoServer installation. On Windows You may be prompted for confirmation to overwrite existing files, confirm the replacement of the files

Moreover, in order for GeoServer to leverage these libraries, the GDAL (binary) libraries must be installed through your host system's OS. Once they are installed, GeoServer will be able to recognize GDAL data types. See below for more information.

Installing GDAL native libraries

The ImageIO-Ext GDAL plugin for geoserver master uses ImageIO-Ext whose latest artifacts can be downloaded from here.

Browse to the native and then gdal directory for the Image IO-Ext download link. Now you should see a list of artifacts that can be downloaded. We need to download two things now:


lib
Tomcat 6.0 ▸ webapps ▸ geoserver ▸ WEB-INF ▸ lib
Organize
Open
New folder
Favorites
Desktop
Downloads
Recent Places
Libraries
Documents
Music
Pictures
Videos
Computer
Local Disk (C:)
Network
Name
Type
cglib-nodep-2.2.jar
com.noelios.restlet.ext.servlet-1.0.8.jar
com.noelios.restlet.ext.simple-1.0.8.jar
com.noelios.restlet-1.0.8.jar
common-2.6.0.jar
commons-beanutils-1.7.0.jar
commons-codec-1.9.jar
commons-collections-3.1.jar
commons-dbcp-1.3.jar
commons-fileupload-1.2.1.jar
commons-httpclient-3.1.jar
commons-io-2.1.jar
commons-jxpath-1.3.jar
commons-lang-2.6.jar
commons-logging-1.1.1.jar
commons-pool-1.5.3.jar
Executable Jar File
batik-parser-1.7.jar Date modified:
Executable Jar File Size:
imageio-ext-1.1.12-jars
geosolutions ▸ Downloads ▸ imageio-ext-1.1.12-jars
Organize
Open
Share with
New folder
Favorites
Desktop
Downloads
Recent Places
Libraries
Documents
Music
Pictures
Videos
Computer
Local Disk (C:)
Network
Name
Type
commons-codec-1.2.jar
commons-io-1.4.jar
commons-lang-2.4.jar
commons-logging-1.1.1.jar
hamcrest-core-1.3.jar
imageio-ext-arcgrid-1.1.12.jar
imageio-ext-gdalarcbinarygrid-1.1.12.jar
imageio-ext-gdalarcgrid-1.1.12.jar
imageio-ext-gdal-bindings-1.9.2.jar
imageio-ext-gdalbsb-1.1.12.jar
imageio-ext-gdaldoq1-1.1.12.jar
imageio-ext-gdaldoq2-1.1.12.jar
imageio-ext-gdaldted-1.1.12.jar
imageio-ext-gdalecw-1.1.12.jar
imageio-ext-gdalecwjp2-1.1.12.jar
Executable Jar File
Move File
There is already a file with the same name in this location.
Click the file you want to keep
Move and Replace
Replace the file in the destination folder with the file you are moving:
commons-logging-1.1.1.jar
commons-logging-1.1.1
(C:\Users\geosolutions\Downloads\imageio-ext-1.1.12-jars)
Size: 59,2 KB
Date modified: 16/07/2015 21:45 (newer)
Don't move
No files will be changed. Leave this file in the destination folder:
commons-logging-1.1.1.jar
commons-logging-1.1.1 (C:\Program Files\Apache Software Foundation\Tomcat 6.0\webapps\geoserver\WEB-INF\lib)
Size: 59,2 KB
Date modified: 12/09/2013 16:31
Move, but keep both files
The file you are moving will be renamed "commons-logging-1.1.1 (2).jar"
Do this for the next 5 conflicts
Skip
Cancel

1. The CRS definitions
2. The native libraries matching the target operating system (more details on picking the right one for your windows installation in the "Extra Steps for Windows Platforms" section)

Let's now install the CRS definitions.

- Click on the "gdal_data.zip" to download the CRS definitions archive.
- Extract this archive on disk and place it in a proper directory on your system.
- Create a GDAL_DATA environment variable to the folder where you have extracted this file. Make also sure that this directory is reachable and readable by the application server process's user.

We now have to install the native libraries.

- Assuming you are using a 64 bit Ubuntu 11 Linux Operating System (for instance), click on the linux folder and then on "gdal192-Ubuntu11-gcc4.5.2-x86_64.tar.gz" to download the native libraries archive (Before doing this, make sure to read and agree with the ECWEULA if you intend to use ECW).
- If you are using a Windows Operating System make sure to download the archive matching your Microsoft Visual C++ Redistributables and your architecture. For example on a 64 bit Windows with 2010 Redistributables, download the gdal-1.9.2-MSVC2010-x64.zip archive
- **Extract the archive on disk and place it in a proper directory on your system.**

> **Warning:** If you are using a version of GDAL more recent than 1.9.2, replace the `imageio-ext-gdal-bindings-1.9.2.jar` file with the equivalent java binding jar (typically named either `gdal.jar` or `imageio-ext-gdal-bindings-*.jar`) included with your GDAL version. If your GDAL version does not include a bindings jar, it was probably not compiled with the java bindings and will not work with GeoServer.

> **Warning:** If you are on Windows, make sure that the GDAL DLL files are on your PATH. If you are on Linux, be sure to set the LD_LIBRARY_PATH environment variable to refer to the folder where the SOs are extracted.

Note: The native libraries contains the GDAL gdalinfo utility which can be used to test whether or not the libs are corrupted. This can be done by browsing to the directory where the libs have been extracted and performing a *gdalinfo* command with the *formats* options that shows all the formats supported. The key element of GDAL support in GeoServer is represented by the JAVA bindings. To test the bindings, the package contains a Java version of the gdalinfo utility inside the "javainfo" folder (a .bat script for Windows and a .sh for Linux), it is very important to run it (again, with the *formats* options) to make sure that the Java bindings are working properly since that is what GeoServer uses. An error message like *Can't load IA 32-bit .dll on a AMD 64-bit platform* in the log files indicates a mixed version of the tools, please go through the installation process again and pick the appropriate versions. More details on troubleshooting are provided in the *Note on running GeoServer as a Service on Windows* section below.

Once these steps have been completed, restart GeoServer. If all the steps have been performed correctly, new data formats will be in the *Raster Data Sources* list when creating a new data store in the *Stores* section as shown here below.

If new formats do not appear in the GUI and you see the following message in the log file:

Fig. 5.92: *GDAL image formats in the list of raster data stores*

it.geosolutions.imageio.gdalframework.GDALUtilities loadGDAL WARNING: Native library load failed.java.lang.UnsatisfiedLinkError: no gdaljni in java.library.path

that means that the installations failed for some reason.

Extra Steps for Windows Platforms

There are a few things to be careful with as well as some extra steps if you are deploying on Windows.

As stated above, we have multiple versions like MSVC2005, MSVC2008 and so on matching the Microsoft Visual C++ Redistributables. Depending on the version of the underlying operating system you'll have to pick up the right one. You can google around for the one you need. Also make sure you download the 32 bit version if you are using a 32 bit version of Windows or the 64 bit version (has a "-x64" suffix in the name of the zip file) if you are running a 64 bit version of Windows. Again, pick the one that matches your infrastructure.

Note on running GeoServer as a Service on Windows

Note that if you downloaded an installed GeoServer as a Windows service you installed the 32 bit version.

Simply deploying the GDAL ImageI/O-Ext native libraries in a location referred by the PATH environment variable (like, as an instance, the JDK/bin folder) doesn't allow GeoServer to leverage on GDAL, when run as a service. As a result, during the service startup, GeoServer log reports this worrisome message:

it.geosolutions.imageio.gdalframework.GDALUtilities loadGDAL WARNING: Native library load failed.java.lang.UnsatisfiedLinkError: no gdaljni in java.library.path

Taking a look at the wrapper.conf configuration file available inside the GeoServer installation (at bin/wrapper/wrapper.conf), there is this useful entry:

Java Library Path (location of Wrapper.DLL or libwrapper.so) wrapper.java.library.path.1=bin/wrapper/lib

To allow the GDAL native DLLs to be loaded, you have two options:

1. Move the native DLLs to the referenced path (bin/wrapper/lib)
2. Add a wrapper.java.library.path.2=path/where/you/deployed/nativelibs entry just after the wrapper.java.library.path1=bin/wrapper/lib line.

Adding support for ECW and MrSID on Windows

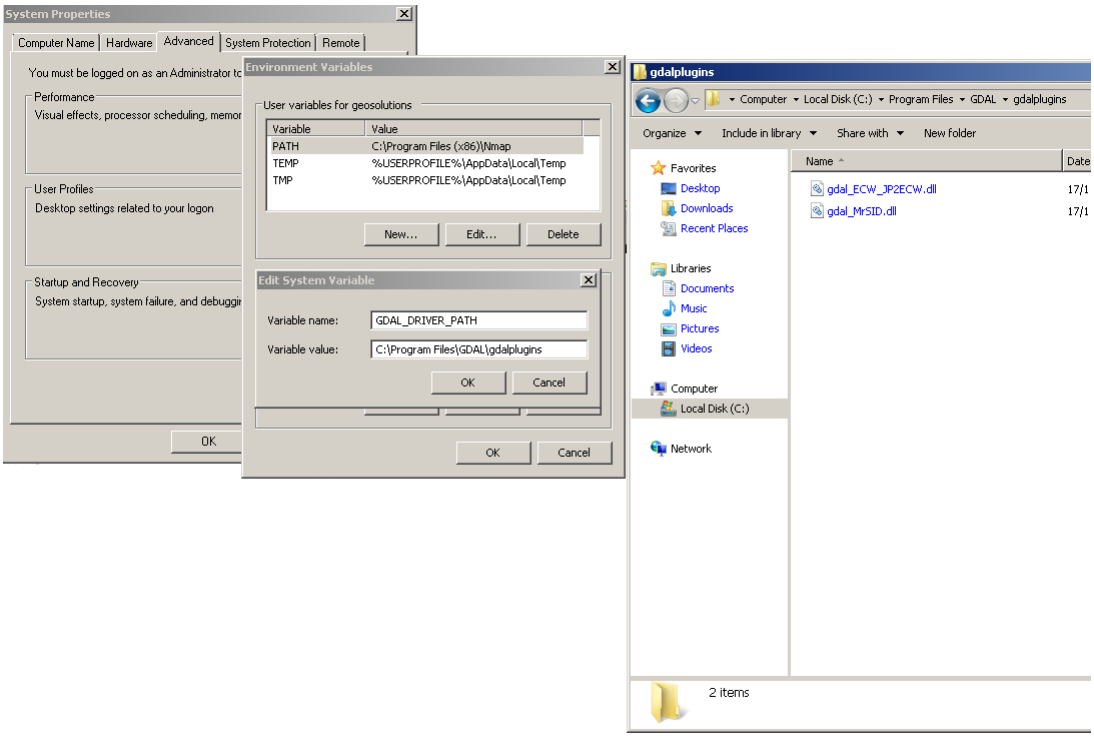
If you are on Windows and you want to add support for ECW and MrSID there is an extra step to perform.

Download and install ECW and MrSID from GeoSolutions site

In the Windows packaging ECW and MrSID are built as plugins hence they are not loaded by default but we need to place their DLLs in a location that is pointed to by the *GDAL_DRIVER_PATH* environment variable. By default the installer place the plugins in C:\Program Files\GDAL\gdalplugins.

GDAL internally uses an environment variable to look up additional drivers (notice that there are a few default places where GDAL will look anyway). For additional information, please see the GDAL wiki.

Restart GeoServer, you should now see the new data sources available



Raster Data Sources

- AIG - Arc/Info Binary Grid (AIG) Coverage Format
- ArcGrid - Arc Grid Coverage Format
- DTED - DTED Coverage Format
- ECW - ECW Coverage Format
- EHdr - EHdr Coverage Format
- ENVIHdr - ENVIHdr Coverage Format
- ERDASImg - Erdas Imagine Coverage Format
- GeoTIFF - Tagged Image File Format with Geographic information
- Gtopo30 - Gtopo30 Coverage Format
- ImageMosaic - Image mosaicking plugin
- JP2ECW - JP2K (ECW) Coverage Format
- MrSID - MrSID Coverage Format
- NITF - NITF Coverage Format
- RPFTOC - RPFTOC Coverage Format
- RST - IDRIS (RST) Coverage Format
- WorldImage - A raster file accompanied by a spatial data file

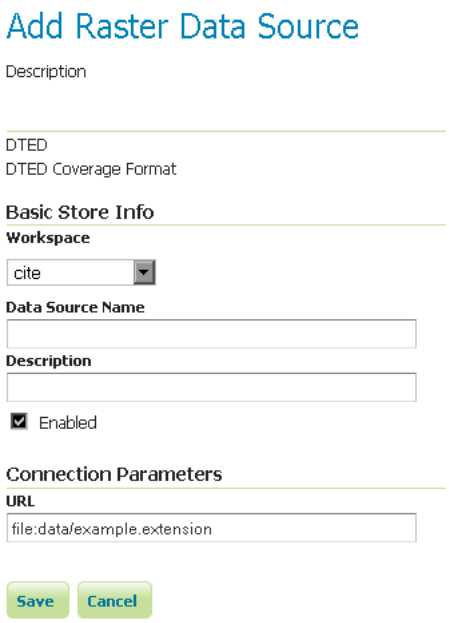


Fig. 5.93: *Configuring a DTED data store*

Configuring a DTED data store

Configuring a EHdr data store

Configuring a ERDASImg data store

Configuring a JP2MrSID data store

Configuring a NITF data store

Supporting vector footprints

Starting with version 2.9.0, GeoServer supports vector footprints. A footprint is a shape used as a mask to hide those pixels that are outside of the mask, hence making that part of the parent image transparent. The currently supported footprint formats are WKB, WKT and Shapefile. By convention, the footprint file should be located in the same directory as the raster data that the footprint applies to.

Note: In the examples of this section and related subsections, we will always use .wkt as extension, representing a WKT footprint, although both .wkb and .shp are supported too.

For example, supposing you have a MrSID file located at `/mnt/storage/data/landsat/N-32-40_2000.sid` to be masked, you just need to place a WKT file on the same folder, as `/mnt/storage/data/landsat/N-32-40_2000.wkt` Note that the footprint needs to have same path and name of the original data file, with .wkt extension.

This is how the sample footprint geometry looks:

Once footprint file has been added, you need to change the FootprintBehavior parameter from None (the default value) to Transparent, from the layer configuration.

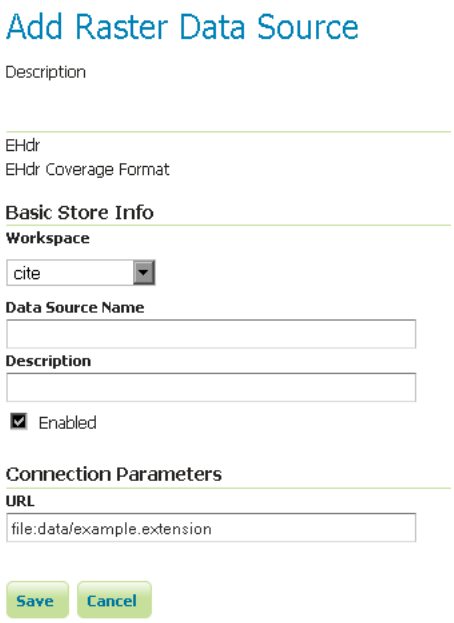


Fig. 5.94: *Configuring a EHdr data store*

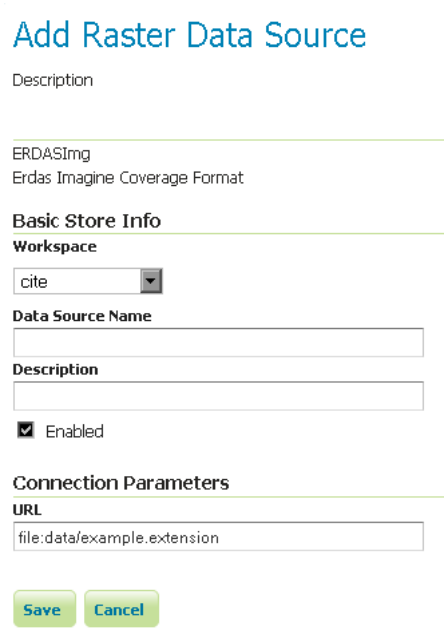


Fig. 5.95: *Configuring a ERDASImg data store*

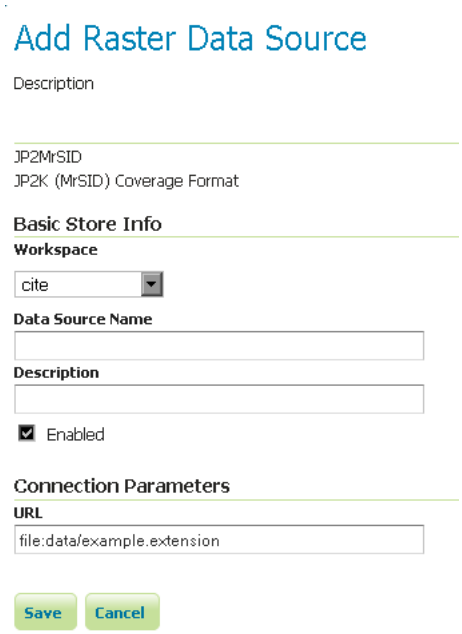


Fig. 5.96: *Configuring a JP2MrSID data store*

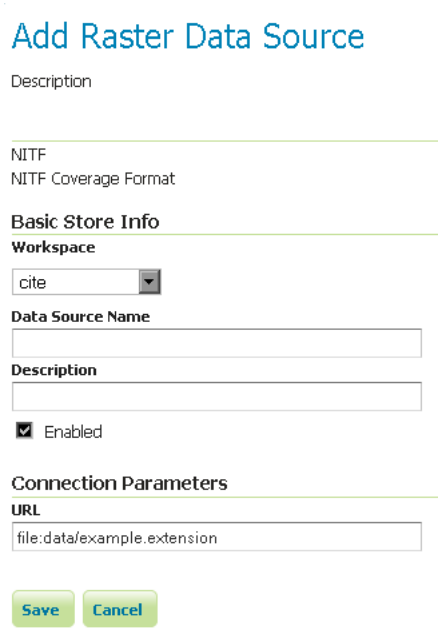


Fig. 5.97: *Configuring a NITF data store*

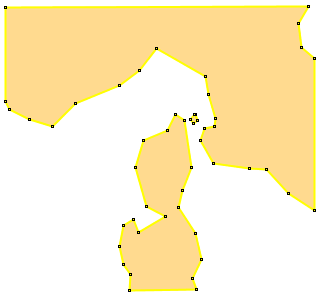


Fig. 5.98: *A sample geometry stored as WKT, rendered on OpenJump*

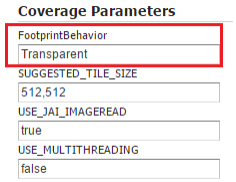


Fig. 5.99: *Setting the FootprintBehavior parameter*

The next image depicts 2 layer previews for the same layer: the left one has no footprint, the right one has a footprint available and FootprintBehavior set to transparent.

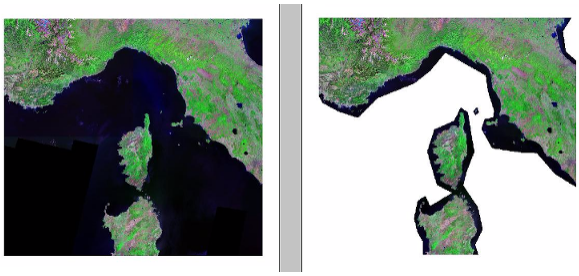


Fig. 5.100: *No Footprint VS FootprintBehavior = Transparent*

External Footprints data directory

As noted above, the footprint file should be placed in the same directory as the raster file. However in some cases this may not be possible. For example, the folder containing the raster data may be read only.

As an alternative, footprint files can be located in a common directory, the **footprints data directory**. The subdirectories and file names under that directory must match the original raster path and file names. The footprints data directory is specified as a Java System Property or an Environment Variable, by setting the *FOOTPRINTS_DATA_DIR* property/variable to the directory to be used as base folder.

Example

Suppose you have 3 raster files with the following paths:

- `/data/raster/charts/nitf/italy_2015.ntf`

- `/data/raster/satellite/ecw/orthofoto_2014.ecw`
- `/data/raster/satellite/landsat/mrsid/N-32-40_2000.sid`

They can be represented by this tree:

```
/data
 \---raster
     +---charts
     |   \---nitf
     |           italy_2015.ntf
     |
     \---satellite
         +---ecw
         |       orthofoto_2014.ecw
         |
         \---landsat
             \---mrsid
                     N-32-40_2000.sid
```

In order to support external footprints you should

1. Create a `/footprints` (as an example) directory on disk
2. Set the `FOOTPRINTS_DATA_DIR=/footprints` variable/property.
3. Replicate the rasters folder hierarchy inside the specified folder, using the full paths.
4. Put the 3 WKT files in the proper locations:

- `/footprints/data/raster/charts/nitf/italy_2015.wkt`
- `/footprints/data/raster/satellite/ecw/orthofoto_2014.wkt`
- `/footprints/data/raster/satellite/landsat/mrsid/N-32-40_2000.wkt`

Which can be represented by this tree:

```
/footprints
 \---data
     \---raster
         +---charts
         |   \---nitf
         |           italy_2015.wkt
         |
         \---satellite
             +---ecw
             |       orthofoto_2014.wkt
             |
             \---landsat
                 \---mrsid
                         N-32-40_2000.wkt
```

Such that, in the end, you will have the following folders hierarchy tree:

```
+---data
|   \---raster
|       +---charts
|       |   \---nitf
|       |           italy_2015.ntf
|       |
|       \---satellite
```

```
|           +---ecw
|           |       orthofoto_2014.ecw
|           |
|           \---landsat
|               \---mrsid
|                       N-32-40_2000.sid
|
\---footprints
    \---data
        \---raster
            +---charts
            |   \---nitf
            |           italy_2015.wkt
            |
            \---satellite
                +---ecw
                |       orthofoto_2014.wkt
                |
                \---landsat
                    \---mrsid
                            N-32-40_2000.wkt
```

Note the parallel mirrored folder hierarchy, with the only differences being a `/footprints` prefix at the beginning of the path, and the change in suffix.

5.3.8 Oracle Georaster

Note: GeoServer does not come built-in with support for Oracle Georaster; it must be installed through an extension. Proceed to *Image Mosaic JDBC* for installation details. This extension includes the support for Oracle Georaster.

Adding an Oracle Georaster data store

Read the geotools documentation for Oracle Georaster Support: http://docs.geotools.org/latest/userguide/library/coverage/oracle.html. After creating the xml config file proceed to the section *Configuring GeoServer* in the *Image Mosaic JDBC Tutorial*

5.3.9 Postgis Raster

Note: GeoServer does not come built-in with support for Postgis raster columns, it must be installed through an extension. Proceed to *Image Mosaic JDBC* for installation details. This extension includes the support for Postgis raster.

Adding an Postgis raster data store

Read the geotools documentation for Postgis raster Support: http://docs.geotools.org/latest/userguide/library/coverage/pgraster.html. After creating the xml config file proceed to the section *Configuring GeoServer* in the *Image Mosaic JDBC Tutorial*

5.3.10 ImagePyramid

Note: GeoServer does not come built-in with support for Image Pyramid; it must be installed through an extension. Proceed to *Installing the ImagePyramid extension* for installation details.

An image pyramid is several layers of an image rendered at various image sizes, to be shown at different zoom levels.

Installing the ImagePyramid extension

1. Download the ImagePyramid extension from the GeoServer download page.

   Warning: Make sure to match the version of the extension to the version of the GeoServer instance!

2. Extract the contents of the archive into the `WEB-INF/lib` directory of the GeoServer installation.

Adding an ImagePyramid data store

Once the extension is properly installed *ImagePyramid* will be an option in the *Raster Data Sources* list when creating a new data store.

Fig. 5.101: *ImagePyramid in the list of raster data stores*

Configuring an ImagePyramid data store

Option	Description
`Workspace`	
`Data Source Name`	
`Description`	
`Enabled`	
`URL`	

5.3.11 Image Mosaic JDBC

Note: GeoServer does not come built-in with support for Image Mosaic JDBC; it must be installed through an extension. Proceed to *Installing the JDBC Image Mosaic extension* for installation details.

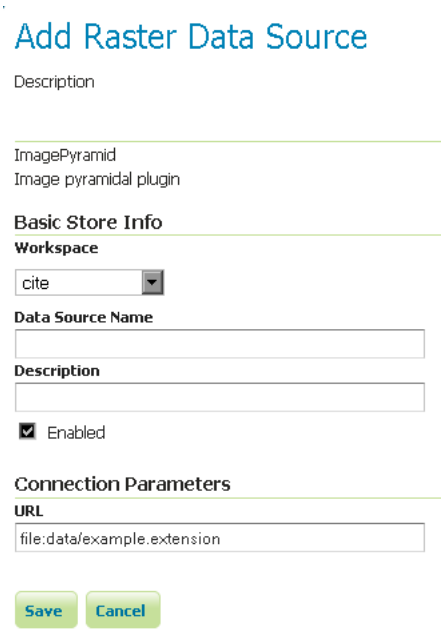


Fig. 5.102: *Configuring an ImagePyramid data store*

Installing the JDBC Image Mosaic extension

1. Download the JDBC Image Mosaic extension from the GeoServer download page.

   Warning: Make sure to match the version of the extension to the version of the GeoServer instance!

2. Extract the contents of the archive into the `WEB-INF/lib` directory of the GeoServer installation.

Adding an Image Mosaic JDBC data store

Once the extension is properly installed *Image Mosaic JDBC* will be an option in the *Raster Data Sources* list when creating a new data store.

Fig. 5.103: *Image Mosaic JDBC in the list of vector data stores*

Configuring an Image Mosaic JDBC data store

For a detailed description, look at the *Tutorial*

5.3.12 Custom JDBC Access for image data

Fig. 5.104: *Configuring an Image Mosaic JDBC data store*

Note: GeoServer does not come built-in with support for Custom JDBC Access; it must be installed through an extension. Proceed to *Image Mosaic JDBC* for installation details. This extension includes the support for Custom JDBC Access.

Adding a coverage based on Custom JDBC Access

This extension is targeted to users having a special database layout for storing their image data or use a special data base extension concerning raster data.

Read the geotools documentation for Custom JDBC Access: http://docs.geotools.org/latest/userguide/library/coverage/jdbc/customized.html.

After developing the custom plugin, package the classes into a jar file and copy it into the `WEB-INF/lib` directory of the geoserver installation.

Create the xml config file and proceed to the section *Configuring GeoServer* in the *Image Mosaic JDBC Tutorial*

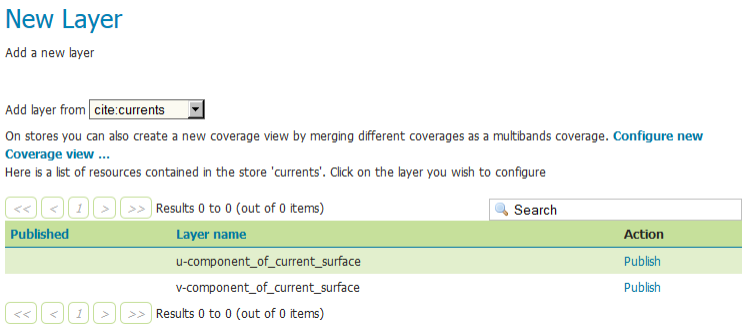
GeoServer provides extensive facilities for controlling how rasters are accessed. These are covered in the following sections.

5.3.13 Coverage Views

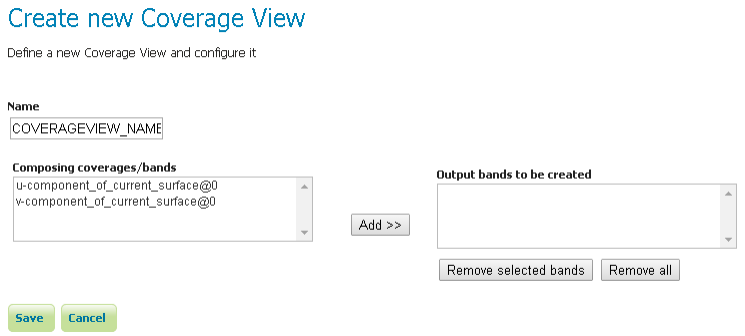
Starting with GeoServer 2.6.0, You can define a new raster layer as a Coverage View. Coverage Views allow defining a View made of different bands originally available inside coverages (either bands of the same coverage or different coverages) of the same Coverage Store.

Creating a Coverage View

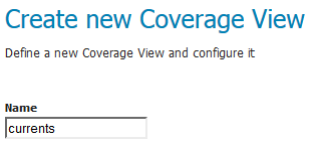
In order to create a Coverage View the administrator invokes the *Create new layer* page. When a Coverage store is selected, the usual list of coverages available for publication appears. A link *Configure new Coverage view...* also appears:



Selecting the *Configure new Coverage view...* link opens a new page where you can configure the coverage view:



The upper text box allows to specify the name to be assigned to this coverage view. (In the following picture we want to create as example, a **currents** view merging together both u and v components of the currents, which are exposed as separated 1band coverages).



Next step is defining the output bands to be put in the coverage view. It is possible to specify which input coverage bands need to be put on the view by selecting them from the *Composing coverages/bands...*.

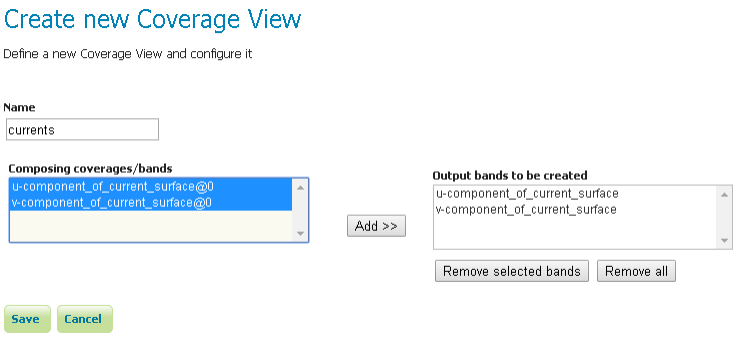
Once selected, they needs to be added to the output bands of the coverage view, using the *add* button.

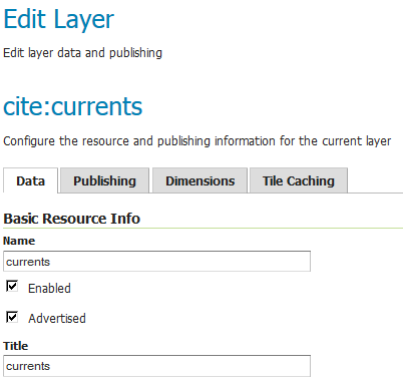
Optionally, is it possible to remove the newly added bands using the *remove* and *remove all* buttons. Once done, clicking on the *save* button will redirect to the standard Layer configuration page.

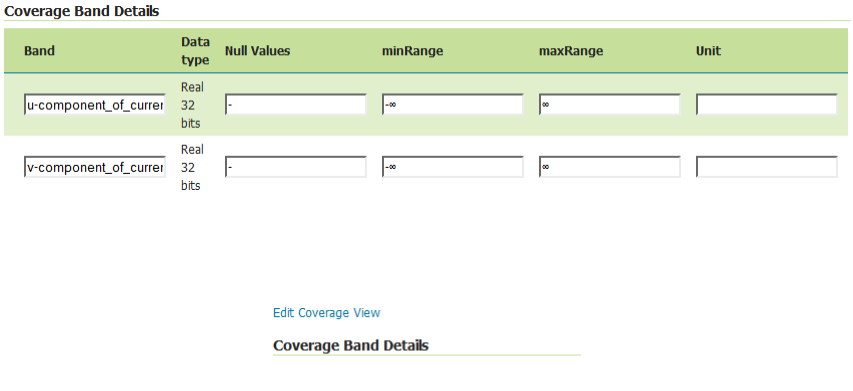
Scrolling down to the end of the page, is it possible to see the bands composing the coverage (and verify they are the one previously selected).

At any moment, the Coverage View can be refined and updated by selecting the *Edit Coverage view...* link available before the Coverage Bands details section.

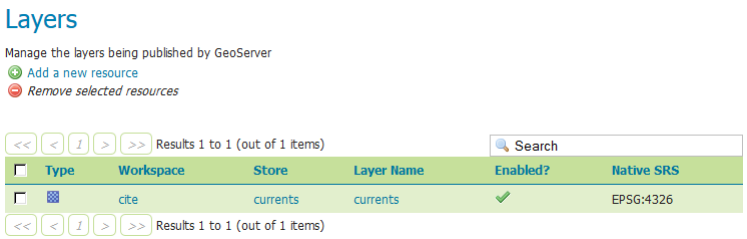

Composing coverages/bands
u-component_of_current_surface@0
v-component_of_current_surface@0
Add >>
Output bands to be created
Remove selected bands
Remove all
Save
Cancel


Create new Coverage View
Define a new Coverage View and configure it
Name
currents
Composing coverages/bands
u-component_of_current_surface@0
v-component_of_current_surface@0
Add >>
Output bands to be created
u-component_of_current_surface
v-component_of_current_surface
Remove selected bands
Remove all
Save
Cancel


Edit Layer
Edit layer data and publishing
cite:currents
Configure the resource and publishing information for the current layer
Data
Publishing
Dimensions
Tile Caching
Basic Resource Info
Name
currents
Enabled
Advertised
Title
currents


Coverage Band Details
Band
Data type
Null Values
minRange
maxRange
Unit
u-component_of_curren
Real 32 bits
-
-∞
∞
v-component_of_curren
Real 32 bits
-
-∞
∞
Edit Coverage View
Coverage Band Details

Once all the properties of the layer have been configured, by selecting the *Save* button, the coverage will be saved in the catalog and it will become visible as a new layer.



Heterogeneous coverage views

In case the various coverages bound in the view have different resolution, the UI will present two extra controls:

The **coverage envelope policy** defines how the bounding box of the output is calculated for metadata purposes. Having different resolutions, the coverages are unlikely to share the same bounding box. The possible values are:

- **Intersect envelopes**: Use the intersection of all input coverage envelopes
- **Union envelopes**: Use the union of all input coverage envelopes

The **coverage resolution policy** defines which target resolution is used when generating outputs:

- **Best**: Use the best resolution available among the chosen bands (e.g., in a set having 60m, 20m and 10m the 10m resolution will be chosen)
- **Worst**: Use the worst resolution available among the chosen bands (e.g., in a set having 60m, 20m and 10m the 60m resolution will be chosen)

The coverage resolution policy is *context sensitive*. Assume the input is a 12 bands Sentinel 2 dataset at three different resolution, 10, 20 and 30 meters, and a false color image is generated by performing a band selection in the SLD. If the policy is *best* and the SLD selects only bands at 20 and 60 meters, the output will be at 20 meters instead of 10 meters.

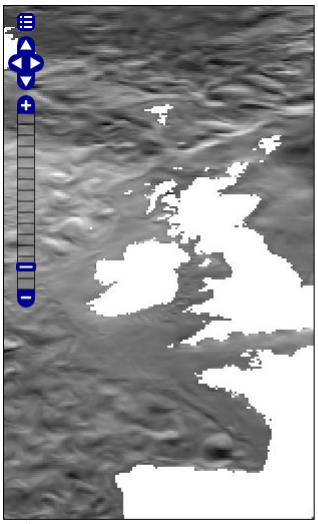
Coverage View in action

A Layer preview of the newly created coverage view will show the rendering of the view. Note that clicking on a point on the map will result into a GetFeatureInfo call which will report the values of the bands composing the coverage view.

5.4 Databases

This section discusses the database data sources that GeoServer can access.

The standard GeoServer installation supports accessing the following databases:



5.4.1 PostGIS

PostGIS is an open source spatial database based on PostgreSQL, and is currently one of the most popular open source spatial databases today.

Adding a PostGIS database

As with all formats, adding a shapefile to GeoServer involves adding a new store to the existing *Stores* through the *Web administration interface*.

Using default connection

To begin, navigate to *Stores* → *Add a new store* → *PostGIS NG*.

Fig. 5.105: *Adding a PostGIS database*

Option	Description
Workspace	Name of the workspace to contain the database. This will also be the prefix of any layer names created from tables in the database.
Data Source Name	Name of the database. This can be different from the name as known to PostgreSQL/PostGIS.
Description	Description of the database/store.
Enabled	Enables the store. If disabled, no data in the database will be served.
dbtype	Type of database. Leave this value as the default.
host	Host name where the database exists.
port	Port number to connect to the above host.
database	Name of the database as known on the host.
schema	Schema in the above database.
user	User name to connect to the database.
passwd	Password associated with the above user.
namespace	Namespace to be associated with the database. This field is altered by changing the workspace name.
max connections	Maximum amount of open connections to the database.
min connections	Minimum number of pooled connections.
fetch size	Number of records read with each interaction with the database.
Connection timeout	Time (in seconds) the connection pool will wait before timing out.
validate connections	Checks the connection is alive before using it.
Loose bbox	Performs only the primary filter on the bounding box. See the section on *Using loose bounding box* for details.
preparedStatements	Enables prepared statements.

When finished, click *Save*.

Using JNDI

GeoServer can also connect to a PostGIS database using JNDI (Java Naming and Directory Interface).

To begin, navigate to *Stores* → *Add a new store* → *PostGIS NG (JNDI)*.

Option	Description
Workspace	Name of the workspace to contain the store. This will also be the prefix of all of the layer names created from the store.
Data Source Name	Name of the database. This can be different from the name as known to PostgreSQL/PostGIS.
Description	Description of the database/store.
Enabled	Enables the store. If disabled, no data in the database will be served.
dbtype	Type of database. Leave this value as the default.
jndiReferenceName	JNDI path to the database.
schema	Schema for the above database.
namespace	Namespace to be associated with the database. This field is altered by changing the workspace name.

When finished, click *Save*.

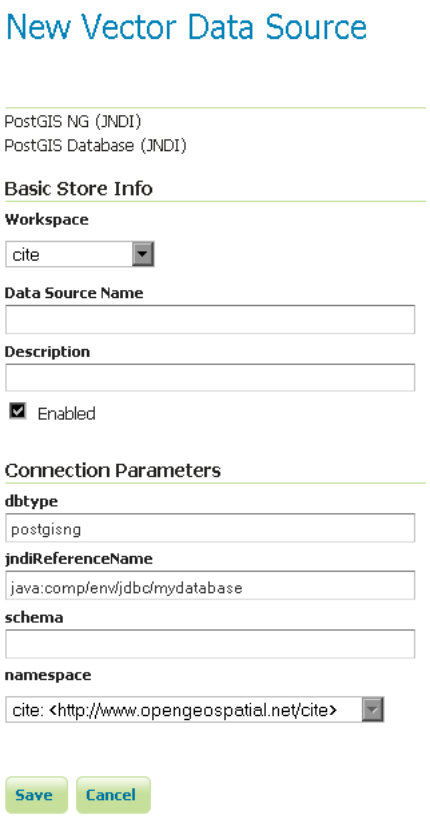


Fig. 5.106: *Adding a PostGIS database (using JNDI)*

Configuring PostGIS layers

When properly loaded, all tables in the database will be visible to GeoServer, but they will need to be individually configured before being served by GeoServer. See the section on *Layers* for how to add and edit new layers.

Using loose bounding box

When the option *loose bbox* is enabled, only the bounding box of a geometry is used. This can result in a significant performance gain, but at the expense of total accuracy; some geometries may be considered inside of a bounding box when they are technically not.

If primarily connecting to this data via WMS, this flag can be set safely since a loss of some accuracy is usually acceptable. However, if using WFS and especially if making use of BBOX filtering capabilities, this flag should not be set.

Publishing a PostGIS view

Publishing a view follows the same process as publishing a table. The only additional step is to manually ensure that the view has an entry in the `geometry_columns` table.

For example consider a table with the schema:

```
my_table( id int PRIMARY KEY, name VARCHAR, the_geom GEOMETRY )
```

Consider also the following view:

```
CREATE VIEW my_view as SELECT id, the_geom FROM my_table;
```

Before this view can be served by GeoServer, the following step is necessary to manually create the `geometry_columns` entry:

```
INSERT INTO geometry_columns VALUES ('','public','my_view','my_geom', 2, 4326, 'POINT
↪' );
```

Performance considerations

GEOS

GEOS (Geometry Engine, Open Source) is an optional component of a PostGIS installation. It is recommended that GEOS be installed with any PostGIS instance used by GeoServer, as this allows GeoServer to make use of its functionality when doing spatial operations. When GEOS is not available, these operations are performed internally which can result in degraded performance.

Spatial indexing

It is strongly recommended to create a spatial index on tables with a spatial component (i.e. containing a geometry column). Any table of which does not have a spatial index will likely respond slowly to queries.

Common problems

Primary keys

In order to enable transactional extensions on a table (for transactional WFS), the table must have a primary key. A table without a primary key is considered read-only to GeoServer.

GeoServer has an option to expose primary key values (to make filters easier). Please keep in mind that these values are only exposed for your convenience - any attempted to modify these values using WFS-T update will be silently ignored. This restriction is in place as the primary key value is used to define the FeatureId. If you must change the FeatureId you can use WFS-T delete and add in a single Transaction request to define a replacement feature.

Multi-line

To insert multi-line text (for use with labeling) remember to use escaped text:

```
INSERT INTO place VALUES (ST_GeomFromText('POINT(-71.060316 48.432044)', 4326), E
↪'Westfield\nTower');
```

5.4.2 H2

Note: GeoServer does not come built-in with support for H2; it must be installed through an extension. Proceed to *Installing the H2 extension* for installation details.

Installing the H2 extension

1. Download the H2 extension from the GeoServer download page.

   Warning: Make sure to match the version of the extension to the version of the GeoServer instance!

2. Extract the contents of the archive into the `WEB-INF/lib` directory of the GeoServer installation.

Adding an H2 data store

Once the extension is properly installed *H2* will be an option in the *Vector Data Sources* list when creating a new data store.

Fig. 5.107: *H2 in the list of vector data stores*

Configuring an H2 data store

Fig. 5.108: *Configuring an H2 data store*

Configuring an H2 data store with JNDI

Other data sources are supplied as GeoServer extensions. Extensions are downloadable modules that add functionality to GeoServer. Extensions are available at the GeoServer download page.

Warning: The extension version must match the version of the GeoServer instance.

5.4.3 ArcSDE

Note: ArcSDE support is not enabled by default and requires the ArcSDE extension to be installed prior to use. Please see the section on *Installing the ArcSDE extension* for details.

ESRI's ArcSDE is a spatial engine that runs on top of a relational database such as Oracle or SQL Server. GeoServer with the ArcSDE extension supports ArcSDE **versions 9.2 and 9.3**. It has been tested with **Oracle 10g** and **Microsoft SQL Server 2000 Developer Edition**. The ArcSDE extension is based on the GeoTools ArcSDE driver and uses the ESRI Java API libraries. See the GeoTools ArcSDE page for more technical details.

There are two types of ArcSDE data that can be added to GeoServer: **vector** and **raster**.

Vector support

ArcSDE provides efficient access to vector layers, ("featureclasses" in ArcSDE jargon), over a number of relational databases. GeoServer can set up featuretypes for registered ArcSDE featureclasses and spatial views. For versioned ArcSDE featureclasses, GeoServer will work on the default database version, for both read and write access.

Transactional support is enabled for featureclasses with a properly set primary key, regardless if the featureclass is managed by a user or by ArcSDE. If a featureclass has no primary key set, it will be available as read-only.

Raster support

ArcSDE provides efficient access to multi-band rasters by storing the raw raster data as database blobs, dividing it into tiles and creating a pyramid. It also allows a compression method to be set for the tiled blob data and an interpolation method for the pyramid resampling.

All the bands comprising a single ArcSDE raster layer must have the same pixel depth, which can be one of 1, 4, 8, 16, and 32 bits per sample for integral data types. For 8, 16 and 32 bit bands, they may be signed or unsigned. 32 and 64 bit floating point sample types are also supported.

ArcSDE rasters may also be color mapped, as long as the raster has a single band of data typed 8 or 16 bit unsigned.

Finally, ArcSDE supports raster catalogs. A raster catalog is a mosaic of rasters with the same spectral properties but instead of the mosaic being precomputed, the rasters comprising the catalog are independent and the mosaic work performed by the application at runtime.

Technical Detail	Status
Compression methods	LZW, JPEG
Number of bands	Any number of bands except for 1 and 4 bit rasters (supported for single-band only).
Bit depth for color-mapped rasters	8 bit and 16 bit
Raster Catalogs	Any pixel storage type

Installing the ArcSDE extension

Warning: Due to licensing requirements, not all files are included with the extension. To install ArcSDE support, it is necessary to download additional files. **Just installing the ArcSDE extension will have no effect.**

GeoServer files

1. Download the ArcSDE extension from the GeoServer download page.

> **Warning:** Make sure to match the version of the extension to the version of the GeoServer instance!

2. Extract the contents of the archive into the `WEB-INF/lib` directory of the GeoServer installation.

Required external files

There are two files that are required but are not packaged with the GeoServer extension:

File	Notes
`jsde_sdk.jar`	Also known as `jsde##_sdk.jar` where `##` is the version number, such as `92` for ArcSDE version 9.2
`jpe_sdk.jar`	Also known as `jpe##_sdk.jar` where `##` is the version number, such as `92` for ArcSDE version 9.2

You should always make sure the `jsde_sdk.jar` and `jpe_sdk.jar` versions match your ArcSDE server version, including service pack, although client jar versions higher than the ArcSDE Server version usually work just fine.

These two files are available on your installation of the ArcSDE Java SDK from the ArcSDE installation media (usually `C:\Program Files\ArcGIS\ArcSDE\lib`). They may also be available on ESRI's website if there's a service pack containing them, but this is not guaranteed. To download these files from ESRI's website:

1. Navigate to http://support.esri.com/index.cfm?fa=downloads.patchesServicePacks.listPatches&PID=66
2. Find the link to the latest service pack for your version of ArcSDE
3. Scroll down to *Installing this Service Pack* → *ArcSDE SDK* → *UNIX* (regardless of your target OS)
4. Download any of the target files (but be sure to match 32/64 bit to your OS)
5. Open the archive, and extract the appropriate JARs.

Note: The JAR files may be in a nested archive inside this archive.

Note: The `icu4j##.jar` may also be on your ArcSDE Java SDK installation folder, but it is already included as part of the the GeoServer ArcSDE extension and is not necessary to install separately.

1. When downloaded, copy the two files to the `WEB-INF/lib` directory of the GeoServer installation.

After all GeoServer files and external files have been downloaded and copied, restart GeoServer.

Adding an ArcSDE vector data store

In order to serve vector data layers, it is first necessary to register the ArcSDE instance as a data store in GeoServer. Navigate to the **New data source** page, accessed from the *Stores* page in the *Web administration interface*. and an option for **ArcSDE** will be in the list of *Vector Data Stores*.

Note: If `ArcSDE` is not an option in the **Feature Data Set Description** drop down box, the extension is not properly installed. Please see the section on *Installing the ArcSDE extension*.

Fig. 5.109: *ArcSDE in the list of data sources*

Configuring an ArcSDE vector data store

The next page contains configuration options for the ArcSDE vector data store. Fill out the form, then click *Save*.

Option	Required?	Description
`Feature Data Set ID`	N/A	The name of the data store as set on the previous page.
`Enabled`	N/A	When this box is checked the data store will be available to GeoServer
`Namespace`	Yes	The namespace associated with the data store.
`Description`	No	A description of the data store.
`server`	Yes	The URL of the ArcSDE instance.
`port`	Yes	The port that the ArcSDE instance is set to listen to. Default is 5151.
`instance`	No	The name of the specific ArcSDE instance, where applicable, depending on the underlying database.
`user`	Yes	The username to authenticate with the ArcSDE instance.
`password`	No	The password associated with the above username for authentication with the ArcSDE instance.
`pool.minConnections`	No	Connection pool configuration parameters. See the *Database Connection Pooling* section for details.
`pool.maxConnections`	No	Connection pool configuration parameters. See the *Database Connection Pooling* section for details.
`pool.timeOut`	No	Connection pool configuration parameters. See the *Database Connection Pooling* section for details.

You may now add featuretypes as you would normally do, by navigating to the *New Layer* page, accessed from the *Layers* page in the *Web administration interface*.

Configuring an ArcSDE vector data store with Direct Connect

ESRI Direct Connect[ESRI DC] allows clients to directly connect to an SDE GEODB 9.2+ without a need of an SDE server instance, and is recommended for high availability environments, as it removes the ArcSDE gateway server as a single point of failure. ESRI DC needs additional platform dependent binary drivers and a working Oracle Client ENVIRONMENT (if connecting to an ORACLE DB). See Properties of a direct

Fig. 5.110: *Configuring a new ArcSDE data store*

connection to an ArcSDE geodatabase in the ESRI ArcSDE documentation for more information on Direct Connect, and Setting up clients for a direct connection for information about connecting to the different databases supported by ArcSDE.

The GeoServer configuration parameters are the same as in the *Configuring an ArcSDE vector data store* section above, with a couple differences in how to format the parameters:

- server: In ESRI Direct Connect Mode a value must be given or the Direct Connect Driver will throw an error, so just put a 'none' there - any String will work!
- port: In ESRI Direct Connect Mode the port has a String representation: *sde:oracle10g*, *sde:oracle11g:/:test*, etc. For further information check ArcSDE connection syntax at the official ArcSDE documentation from ESRI.
- instance: In ESRI Direct Connect Mode a value must be given or the Direct Connect Driver will throw an error, so just put a 'none' there - any String will work!
- user: The username to authenticate with the geo database.
- password: The password associated with the above username for authentication with the geo database.

Note: Be sure to assemble the password like: password@<Oracle Net Service name> for Oracle

---

You may now add featuretypes as you would normally do, by navigating to the New Layer page, accessed from the Layers page in the Web Administration Interface.

Adding an ArcSDE vector data store with JNDI

Configuring an ArcSDE vector data store with JNDI

Adding an ArcSDE raster coveragestore

In order to serve raster layers (or coverages), it is first necessary to register the ArcSDE instance as a store in GeoServer. Navigate to the **Add new store** page, accessed from the *Stores* page in the *Web administration interface* and an option for **ArcSDE Raster Format** will be in list.

Note: If `ArcSDE Raster Format` is not an option in the **Coverage Data Set Description** drop down box, the extension is not properly installed. Please see the section on *Installing the ArcSDE extension*.

Fig. 5.111: *ArcSDE Raster in the list of data sources*

Configuring an ArcSDE raster coveragestore

The next page contains configuration options for the ArcSDE instance. Fill out the form, then click *Save*.

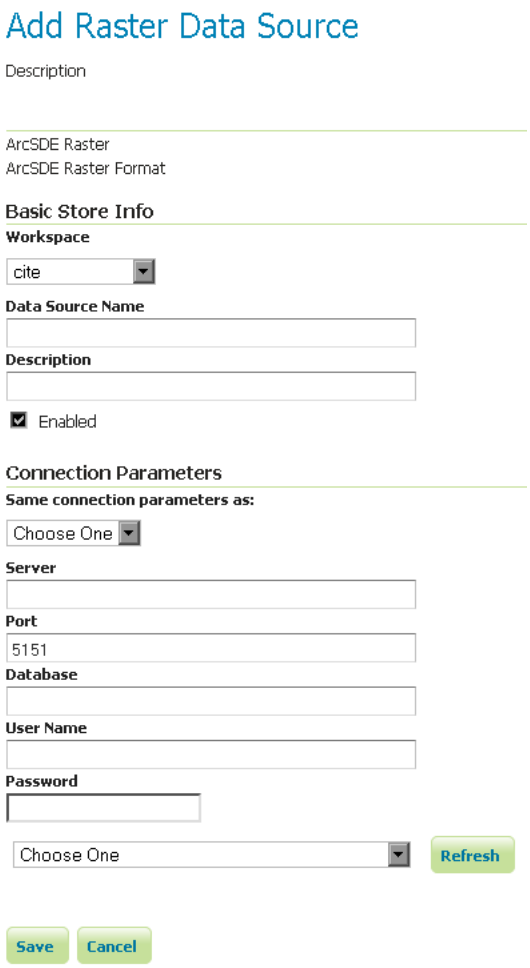


Fig. 5.112: *Configuring a new ArcSDE coveragestore*

Option	Required?	Description
`Coverage Data Set ID`	N/A	The name of the coveragestore as set on the previous page.
`Enabled`	N/A	When this box is checked the coveragestore will be available to GeoServer.
`Namespace`	Yes	The namespace associated with the coveragestore.
`Type`	No	The type of coveragestore. Leave this to say `ArcSDE Raster`.
`URL`	Yes	The URL of the raster, of the form `sde://<user>:<pwd>@<server>/#<tableName>`.
`Description`	No	A description of the coveragestore.

You may now add coverages as you would normally do, by navigating to the **Add new layer** page, accessed from the *Layers* page in the *Web administration interface*.

5.4.4 DB2

Note: GeoServer does not come built-in with support for DB2; it must be installed through an extension. Proceed to *Installing the DB2 extension* for installation details.

The IBM DB2 UDB database is a commercial relational database implementing ISO SQL standards and is similar in functionality to Oracle, SQL Server, MySQL, and PostgreSQL. The DB2 Spatial Extender is a no-charge licensed feature of DB2 UDB which implements the OGC specification "Simple Features for SQL using types and functions" and the ISO "SQL/MM Part 3 Spatial" standard.

A trial copy of DB2 UDB and Spatial Extender can be downloaded from: http://www-306.ibm.com/software/data/db2/udb/edition-pde.html . There is also an "Express-C" version of DB2, that is free, comes with spatial support, and has no limits on size. It can be found at: http://www-306.ibm.com/software/data/db2/express/download.html

Installing the DB2 extension

Warning: Due to licensing requirements, not all files are included with the extension. To install DB2 support, it is necessary to download additional files. **Just installing the DB2 extension will have no effect.**

GeoServer files

1. Download the DB2 extension from the GeoServer download page.

   Warning: Make sure to match the version of the extension to the version of the GeoServer instance!

2. Extract the contents of the archive into the `WEB-INF/lib` directory of the GeoServer installation.

Required external files

There are two files that are required but are not packaged with the GeoServer extension: `db2jcc.jar` and `db2jcc_license_cu.jar`. These files should be available in the `java` subdirectory of your DB2 installation directory. Copy these files to the `WEB-INF/lib` directory of the GeoServer installation.

After all GeoServer files and external files have been downloaded and copied, restart GeoServer.

Adding a DB2 data store

When properly installed, *DB2* will be an option in the *Vector Data Sources* list when creating a new data store.

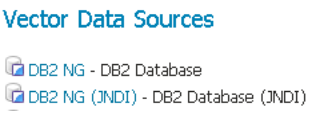


Fig. 5.113: *DB2 in the list of raster data stores*

Configuring a DB2 data store

Configuring a DB2 data store with JNDI

Notes on usage

DB2 schema, table, and column names are all case-sensitive when working with GeoTools/GeoServer. When working with DB2 scripts and the DB2 command window, the default is to treat these names as upper-case unless enclosed in double-quote characters.

5.4.5 MySQL

Note: GeoServer does not come built-in with support for MySQL; it must be installed through an extension. Proceed to *Installing the MySQL extension* for installation details.

Warning: Currently the MySQL extension is unmaintained and carries unsupported status. While still usable, do not expect the same reliability as with other extensions.

MySQL is an open source relational database with some limited spatial functionality.

Installing the MySQL extension

1. Download the MySQL extension from the GeoServer download page.

   Warning: Make sure to match the version of the extension to the version of the GeoServer instance!

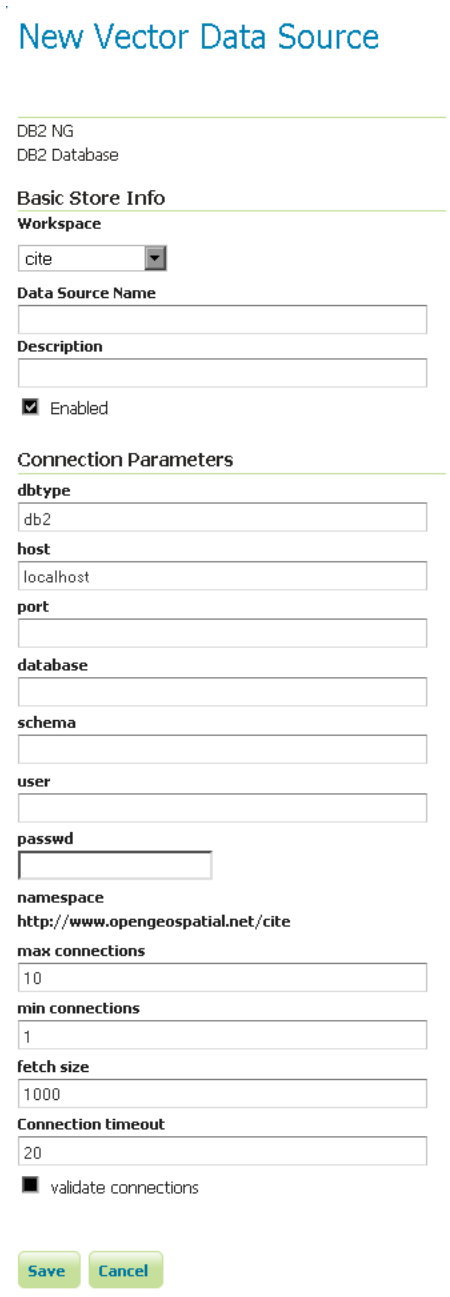


Fig. 5.114: *Configuring a DB2 data store*

2. Extract the contents of the archive into the `WEB-INF/lib` directory of the GeoServer installation.

Adding a MySQL database

Once the extension is properly installed `MySQL` will show up as an option when creating a new data store.

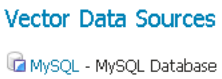


Fig. 5.115: *MySQL in the list of data sources*

Configuring a MySQL data store

`host`	The mysql server host name or ip address.
`port`	The port on which the mysql server is accepting connections.
`database`	The name of the database to connect to.
`user`	The name of the user to connect to the mysql database as.
`password`	The password to use when connecting to the database. Left blank for no password.
`max connections` `min connections` `validate connections`	Connection pool configuration parameters. See the *Database Connection Pooling* section for details.

5.4.6 Oracle

Note: GeoServer does not come built-in with support for Oracle; it must be installed through an extension. Proceed to *Installing the Oracle extension* for installation details.

Oracle Spatial and Locator are the spatial components of Oracle. **Locator** is provided with all Oracle versions, but has limited spatial functions. **Spatial** is Oracle's full-featured spatial offering, but requires a specific license to use.

Installing the Oracle extension

Warning: Due to licensing requirements, not all files are included with the extension. To install Oracle support, it is necessary to download additional files.

1. Download the Oracle extension from the GeoServer download page.

   Warning: Make sure to match the version of the extension to the version of the GeoServer instance!

Fig. 5.116: *Configuring a MySQL data store*

2. Extract the contents of the archive into the `WEB-INF/lib` directory of the GeoServer installation.
3. Get the Oracle JDBC driver from either your Oracle installation (e.g. `ojdbc6.jar`, `ojdbc7.jar`) or download them from the Oracle JDBC driver distribution page

Consider replacing the Oracle JDBC driver

The Oracle data store zip file comes with `ojdbc4.jar`, an old, Oracle 10 compatible JDBC driver that normally works fine with 11g as well. However, minor glitches have been observed with 11g (issues computing layer bounds when session initiation scripts are in use) and the driver has not been tested with 12i.

If you encounter functionality or performance issues it is advised to remove this driver and download the latest version from the Oracle web site.

Adding an Oracle datastore

Once the extension is properly installed *Oracle* appears as an option in the *Vector Data Sources* list when creating a new data store.

Fig. 5.117: *Oracle in the list of data sources*

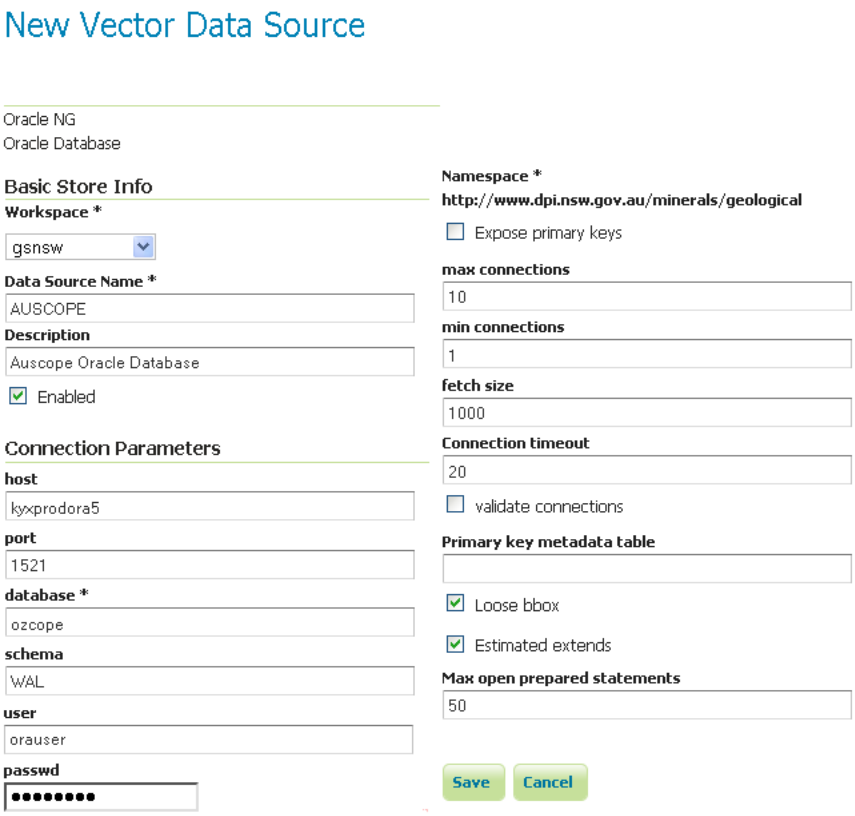


Fig. 5.118: *Configuring an Oracle datastore*

Configuring an Oracle datastore

Option	Description
`host`	The Oracle server host name or IP address.
`port`	The port on which the Oracle server is accepting connections (often this is port 1521).
`database`	The name of the database to connect to. By default this is interpreted as a SID name. To connect to a Service, prefix the name with a `/`.
`schema`	The database schema to access tables from. Setting this value greatly increases the speed at which the data store displays its publishable tables and views, so it is advisable to set this.
`user`	The name of the user to use when connecting to the database.
`password`	The password to use when connecting to the database. Leave blank for no password.
`max connections` `min connections` `fetch size` `Connection timeout` `validate connections`	Connection pool configuration parameters. See *Database Connection Pooling* for details.
`Loose bbox`	Controls how bounding box filters are made against geometries in the database. See the *Using loose bounding box* section below.
`Metadata bbox`	Flag controlling the use of MDSYS.USER_SDO_GEOM_METADATA or MDSYS.ALL_SDO_GEOM_METADATA table for bounding box calculations, this brings a better performance if the views access is fast and the bounds are configured right in the tables default is false

Connecting to an Oracle cluster

In order to connect to an Oracle RAC one can use an almost full JDBC url as the `database`, provided it starts with `(` it will be used verbatim and options "host" and "port" will be ignored. Here is an example "database" value used to connect to an Oracle RAC:

```
(DESCRIPTION=(LOAD_BALANCE=on)(ADDRESS=(PROTOCOL=TCP)(HOST=host1)␣
↪(PORT=1521))(ADDRESS=(PROTOCOL=TCP)(HOST=host2) (PORT=1521))(CONNECT_DATA=(SERVICE_
↪NAME=service)))
```

More information about this syntax can be found in the Oracle documentation.

Connecting to a SID or a Service

Recent versions of Oracle support connecting to a database via either a SID name or a Service name. A SID connection descriptor has the form: `host:port:database`, while a Service connection descriptor has the format `host:port/database`. GeoServer uses the SID form by default. To connect via a Service, prefix the `database` name configuration entry with a `/`.

Connecting to database through LDAP

For instance if you want to establish a connection with the jdbc thin driver through LDAP, you can use following connect string for the input field `database ldap://[host]:[Port]/[db],cn=OracleContext,dc=[oracle_ldap_context]`.

If you are using referrals, enable it by placing a jndi.properties file in geoserver's CLASSPATH, which is in geoserver/WEB-INF/classes. This property file contains:

java.naming.referral=follow

Using loose bounding box

When the `Loose bbox` option is set, only the bounding box of database geometries is used in spatial queries. This results in a significant performance gain. The downside is that some geometries may be reported as intersecting a BBOX when they actually do not.

If the primary use of the database is through the *Web Map Service (WMS)* this flag can be set safely, since querying more geometries does not have any visible effect. However, if using the *Web Feature Service (WFS)* and making use of BBOX filtering capabilities, this flag should not be set.

Using the geometry metadata table

The Oracle data store by default looks at the `MDSYS.USER_SDO*` and `MDSYS.ALL_SDO*` views to determine the geometry type and native SRID of each geometry column. Those views are automatically populated with information about the geometry columns stored in tables that the current user owns (for the `MDSYS.USER_SDO*` views) or can otherwise access (for the `MDSYS.ALL_SDO*` views).

There are a few issues with this strategy:

- if the connection pool user cannot access the tables (because *impersonation* is used) the MDSYS views will be empty, making it impossible to determine both the geometry type and the native SRID
- the geometry type can be specified only while building the spatial indexes, as an index constraint. However such information is often not included when creating the indexes
- the views are populated dynamically based on the current user. If the database has thousands of tables and users the views can become very slow

Starting with GeoServer 2.1.4 the administrator can address the above issues by manually creating a geometry metadata table describing each geometry column. Its presence is indicated via the Oracle datastore connection parameter named *Geometry metadata table* (which may be a simple table name or a schema-qualified one). The table has the following structure (the table name is flexible, just specify the one chosen in the data store connection parameter):

```
CREATE TABLE GEOMETRY_COLUMNS(
   F_TABLE_SCHEMA VARCHAR(30) NOT NULL,
   F_TABLE_NAME VARCHAR(30) NOT NULL,
   F_GEOMETRY_COLUMN VARCHAR(30) NOT NULL,
   COORD_DIMENSION INTEGER,
   SRID INTEGER NOT NULL,
   TYPE VARCHAR(30) NOT NULL,
   UNIQUE(F_TABLE_SCHEMA, F_TABLE_NAME, F_GEOMETRY_COLUMN),
   CHECK(TYPE IN ('POINT','LINE', 'POLYGON', 'COLLECTION', 'MULTIPOINT', 'MULTILINE',
↪'MULTIPOLYGON', 'GEOMETRY') ));
```

When the table is present the store first searches it for information about each geometry column to be classified, and falls back on the MDSYS views only if the table does not contain any information.

Configuring an Oracle database with JNDI

See *Setting up a JNDI connection pool with Tomcat* for a guide on setting up an Oracle connection using JNDI.

5.4.7 Microsoft SQL Server and SQL Azure

Note: GeoServer does not come built-in with support for SQL Server; it must be installed through an extension. Proceed to *Installing the SQL Server extension* for installation details.

Microsoft's SQL Server is a relational database with spatial functionality. SQL Azure is the database option provided in the Azure cloud solution which is in many respects similar to SQL Server 2008.

Supported versions

The extension supports SQL Server 2008 and SQL Azure.

Installing the SQL Server extension

Warning: Due to licensing requirements, not all files are included with the extension. To install SQL Server support, it is necessary to download additional files.

GeoServer files

1. Download the SQL Server extension from the GeoServer download page.

   Warning: Make sure to match the version of the extension to the version of the GeoServer instance!

2. Extract the contents of the archive into the `WEB-INF/lib` directory of the GeoServer installation.
3. Restart the GeoServer to load the extension.

Microsoft files

1. Navigate to the download page for Microsoft JDBC Drivers for SQL Server.
2. Extract the contents of the archive.
3. Copy the file `sqljdbc4.jar` to the `WEB-INF/lib` directory of the GeoServer installation.
4. If GeoServer is installed on Windows, additionally copy `auth\x86\sqljdbc_auth.dll` and `xa\x86\sqljdbc_xa.dll` to `C:\Windows\System32`.

Adding a SQL Server database

Once the extension is properly installed `SQL Server` will show up as an option when creating a new data store.

Fig. 5.119: *SQL Server in the list of vector data sources*

Configuring a SQL Server data store

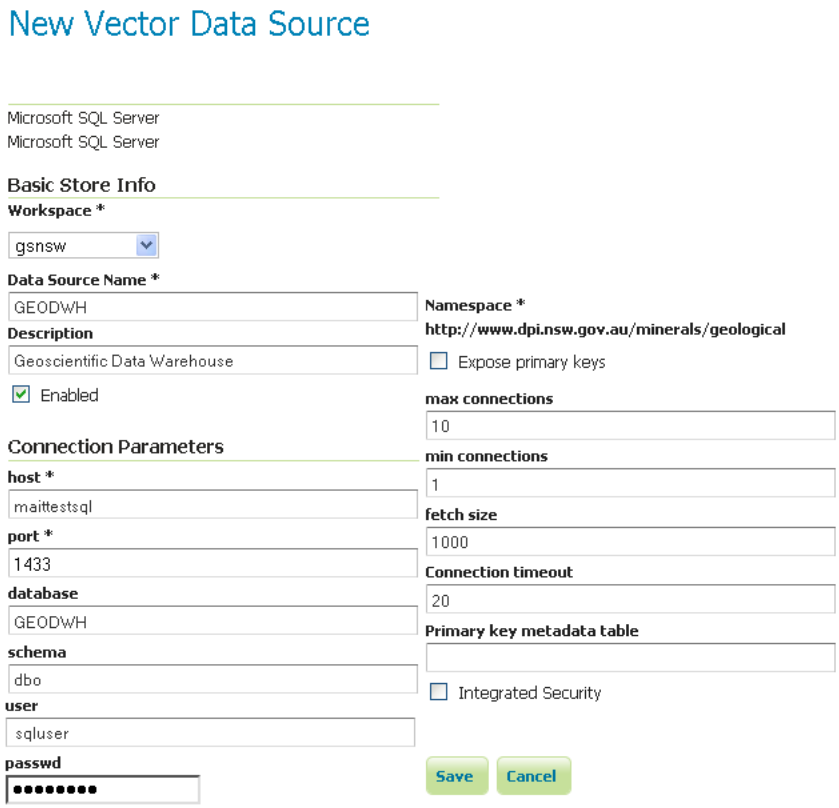


Fig. 5.120: *Configuring a SQL Server data store*

host	The sql server instance host name or ip address, only. Note that server\instance notation is not accepted - specify the port below, instead, if you have a non-default instance.
port	The port on which the SQL server instance is accepting connections. See the *note* below.
database	The name of the database to connect to. Might be left blank if the user connecting to SQL Server has a "default database" set in the user configuration
schema	The database schema to access tables from (optional).
user	The name of the user to connect to the oracle database as.
password	The password to use when connecting to the database. Leave blank for no password.
max connections min connections	Connection pool configuration parameters. See the *Database Connection Pooling* section for details. If you are connecting to SQL Azure make sure to set the validate connections flag as SQL Azure closes inactive connections after a very short delay.

Determining the port used by the SQL Server instance

You can determine the port in use by connecting to your SQL server instance using some other software, and then using **netstat** to display details on network connections. In the following example on a Windows PC, the port is 2646

```
C:\>netstat -a | find "sql1"
TCP    DPI908194:1918   maittestsql1.dpi.nsw.gov.au:2646   ESTABLISHED
```

Using the geometry metadata table

The SQL server data store can determine the geometry type and native SRID of a particular column only by data inspection, by looking at the first row in the table. Of course this is error prone, and works only if there is data in the table. The administrator can address the above issue by manually creating a geometry metadata table describing each geometry column. Its presence is indicated via the SQL Server datastore connection parameter named *Geometry metadata table* (which may be a simple table name or a schema-qualified one). The table has the following structure (the table name is flexible, just specify the one chosen in the data store connection parameter):

```
CREATE TABLE GEOMETRY_COLUMNS(
   F_TABLE_SCHEMA VARCHAR(30) NOT NULL,
   F_TABLE_NAME VARCHAR(30) NOT NULL,
   F_GEOMETRY_COLUMN VARCHAR(30) NOT NULL,
   COORD_DIMENSION INTEGER,
   SRID INTEGER NOT NULL,
   TYPE VARCHAR(30) NOT NULL,
   UNIQUE(F_TABLE_SCHEMA, F_TABLE_NAME, F_GEOMETRY_COLUMN),
   CHECK(TYPE IN ('POINT','LINE', 'POLYGON', 'COLLECTION', 'MULTIPOINT', 'MULTILINE',
↪'MULTIPOLYGON', 'GEOMETRY') ));
```

When the table is present the store first searches it for information about each geometry column to be classified, and falls back on data inspection only if the table does not contain any information.

5.4.8 Teradata

Note: Teradata database support is not enabled by default and requires the Teradata extension to be installed prior to use. Please see the section on *Installing the Teradata extension* for details.

The Teradata Database is a commercial relational database (RDBMS) that specializes in parallel processing and scalability. From version 12.0, Teradata has added geospatial support, closely following the SQL/MM standard (SQL Multimedia and Applications Packages). Geospatial support was available through an add-on in version 12.0 and became standard in version 13.0.

GeoServer connects to a Teradata database via JDBC.

For more information on Teradata and the Teradata Database system, please go to http://www.teradata.com.

Compatibility

The GeoServer Teradata extension is compatible with GeoServer 2.1.1 and higher. GeoServer can connect to Teradata databases version 12.0 or higher. Version 12.0 of the Teradata Database requires the optional geospatial extension to be installed.

Read/write access

The Teradata datastore in GeoServer supports full transactional capabilities, including feature creation, editing, and deleting.

To support editing, a table must have one of the following:

- a primary key
- a unique primary index
- an identity (sequential) column

Note: It is not recommended to solely use an identity column, as spatial index triggers are not supported when referencing an identity column. See the section on Spatial Indexes for more details.

Query Banding

The GeoServer Teradata extension supports Query Banding. Query Banding is a feature which allows any application to associate context information with each query it issues to the database. In practice this can be used for purposes of workload management (i.e. request prioritization), debugging, and logging.

GeoServer sends the following information as part of a standard request:

- Name of application (i.e. GeoServer)
- Authenticated username (if set up)
- Hostname (if available)
- Type of statement (i.e. “SELECT”, “INSERT”, “DELETE”)

It is not possible to modify this information from within GeoServer.

Spatial indexes

GeoServer will read from a spatial index if its exists. The convention for a spatial index table name is:

```
[TABLENAME]_[GEOMETRYCOLUMN]_idx
```

So for a layer called "STATES" with a geometry column called "GEOM", the index table should be called *STATES_GEOM_idx*.

Warning: Make sure to match the case of all tables and columns. If the geometry column is called "GEOM" (upper case) and the index created is called *STATES_geom_idx* (lower case), the index will not be properly linked to the table.

This index table should contain two columns:

- A column that maps to the primary key of the spatial data table
- The tessellation cell ID (cellid)

The tessellation cell ID is the ID of the cell where that feature is contained.

Geometry column

As per the SQL/MM standard, in order to make a Teradata table spatially enabled, an entry needs to be created for that table in the `geometry_columns` table. This table is stored, like all other spatially-related tables, in the SYSSPATIAL database.

Tessellation

Tessellation is the name of Teradata's spatial index. In order to activate tessellation for a given layer, an entry (row) needs to be placed in the `SYSSPATIAL.tessellation` table. This table should have the following schema:

Table name	Type	Description
F_TABLE_SCHEMA	varchar	Name of the spatial database/schema containing the table
F_TABLE_NAME	varchar	Name of the spatial table
F_GEOMETRY_COLUMN	varchar	Column that contains the spatial data
U_XMIN	float	Minimum X value for the tessellation universe
U_YMIN	float	Minimum Y value for the tessellation universe
U_XMAX	float	Maximum X value for the tessellation universe
U_YMAX	float	Maximum Y value for the tessellation universe
G_NX	integer	Number of X grids
G_NY	integer	Number of Y grids
LEVELS	integer	Number of levels in the grid
SCALE	float	Scale value for the grid
SHIFT	float	Shift value for the grid

Warning: The tessellation table values are case sensitive and so must match the case of the tables and columns.

Installing the Teradata extension

Teradata database support is not enabled by default and requires the GeoServer Teradata extension to be installed prior to use. In addition to this extension, an additional artifact will need to be downloaded from the Teradata website.

GeoServer artifacts

1. Download the Teradata extension from the download page that matches your version of GeoServer. The extension is listed at the bottom of the download page under *Extensions*.

   Warning: Make sure to match the version of the extension to the version of GeoServer!

2. Extract the contents of the archive into the `WEB-INF/lib` directory of the GeoServer installation.

Teradata artifacts

In addition to the GeoServer artifacts, it is also necessary to download the Teradata JDBC driver. This file cannot be redistributed and so must be downloaded directly from the Teradata website.

1. Download the Teradata JDBC driver at https://downloads.teradata.com/download/connectivity/jdbc-driver.

   Note: You will need to log in to Teradata's site in order to download this artifact.

2. Extract the contents of the archive into the `WEB-INF/lib` directory of the GeoServer installation.

When all files have been downloaded and extracted, restart GeoServer. To verify that the installation was successful, see the section on *Adding a Teradata datastore*.

Note: The full list of files required are:

- `gt-jdbc-teradata-<version>.jar`
- `tdgssconfig.jar`
- `terajdbc4.jar`

Adding a Teradata datastore

Once the extension has been added, it will now be possible to load an existing Teradata database as a store in GeoServer. In the *Web administration interface*, click on *Stores* then go to *Add a new Store*. You will see a option, under *Vector Data Stores*, for *Teradata*. Select this option.

Note: If you don't Teradata in this list, the extension has not been installed properly. Please ensure that the steps in the *Installing the Teradata extension* have been followed correctly.

On the next screen, enter in the details on how to connect to the Teradata database. You will need to include the following information:

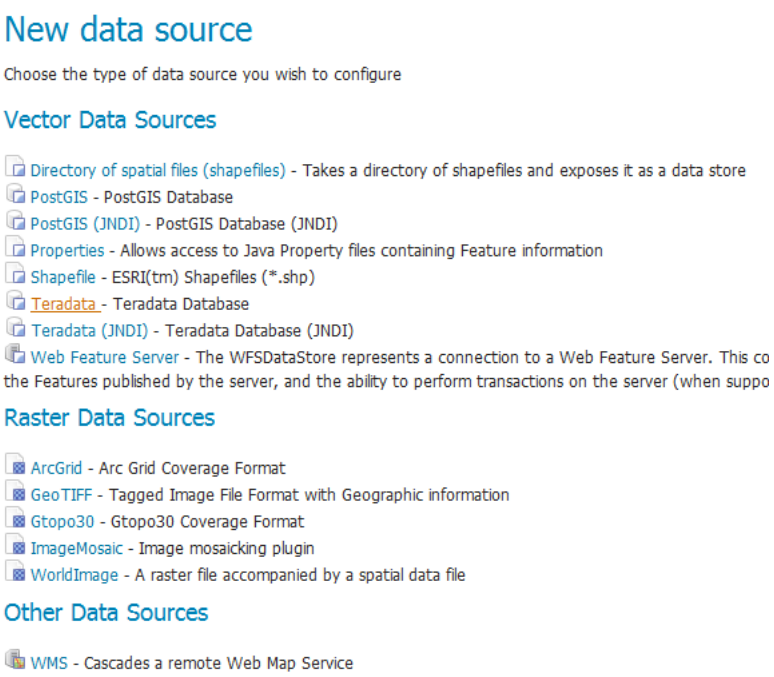
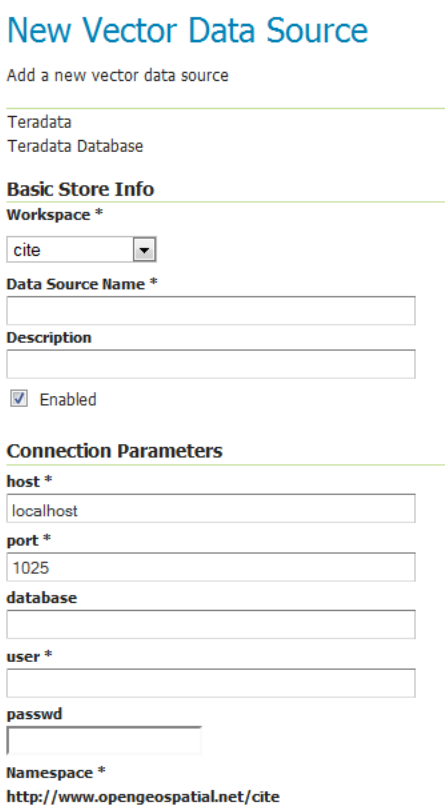


Fig. 5.121: *Teradata in the list of readable stores*

Option	Description
Workspace	Name of the workspace to contain the database. This will also be the prefix of any layers server from tables in the database.
Data Source Name	Name of the database in GeoServer. This can be different from the name of the Teradata database, if desired.
Description	Description of the database/store.
Enabled	Enables the store. If disabled, no layers from the database will be served.
host	Host name where the database exists. Can be a URL or IP address.
port	Port number on which to connect to the above host.
database	Name of the Teradata database.
user	User name to connect to use to connect to the database.
passwd	Password associated with the above user.
namespace	Namespace to be associated with the database. This field is altered automatically by the above Workspace field.
Expose primary keys	Exposes primary key as a standard attribute.
max connections	Maximum amount of open/pooled connections to the database.
min connections	Minimum number of open/pooled connections.
fetch size	Number of records read with each interaction with the database.
Connection timeout	Time (in seconds) the connection pool will wait before timing out.
validate connections	Checks the connection is alive before using it.
Primary key metadata table	Name of primary key metadata table to use if unable to determine the primary key of a table.
Loose bbox	If checked, performs only the primary filter on the bounding box.
tessellationTable	The name of the database table that contains the tessellations
estimatedBounds	Enables using the geometry_columns/tessellation table bounds as an estimation instead of manual calculation.
Max open prepared statements	The maximum number of prepared statements.

When finished, click *Save*.



Using JNDI

GeoServer can also connect to a Teradata database using JNDI (Java Naming and Directory Interface).

To begin, in the *Web administration interface*, click on *Stores* then go to *Add a new Store*. You will see a option, under *Vector Data Stores*, for *Teradata (JNDI)*. Select this option.

On the next screen, enter in the details on how to connect to the Teradata database. You will need to include the following information:

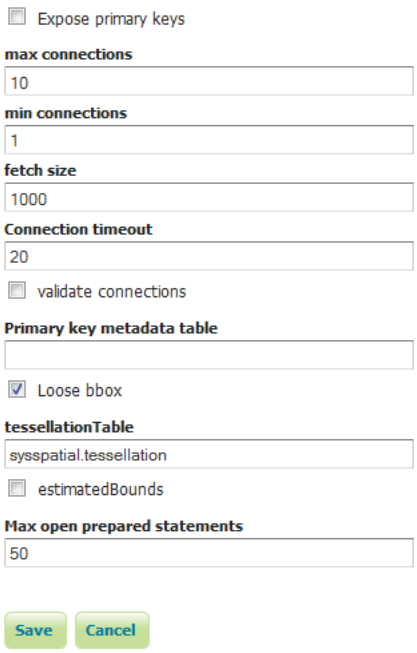


Fig. 5.122: *Adding a Teradata store*

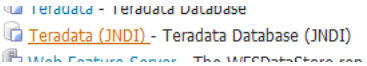


Fig. 5.123: *Teradata (JNDI) in the list of readable stores*

Option	Description
Workspace	Name of the workspace to contain the database. This will also be the prefix of any layers server from tables in the database.
Data Source Name	Name of the database in GeoServer. This can be different from the name of the Teradata database, if desired.
Description	Description of the database/store.
Enabled	Enables the store. If disabled, no layers from the database will be served.
jndiReferenceName	JNDI path to the database.
schema	Schema for the above database.
namespace	Namespace to be associated with the database. This field is altered by changing the workspace name.
Expose primary keys	Exposes primary key as a standard attribute.
Primary key metadata table	Name of primary key metadata table to use if unable to determine the primary key of a table.
Loose bbox	If checked, performs only the primary filter on the bounding box.

When finished, click *Save*.

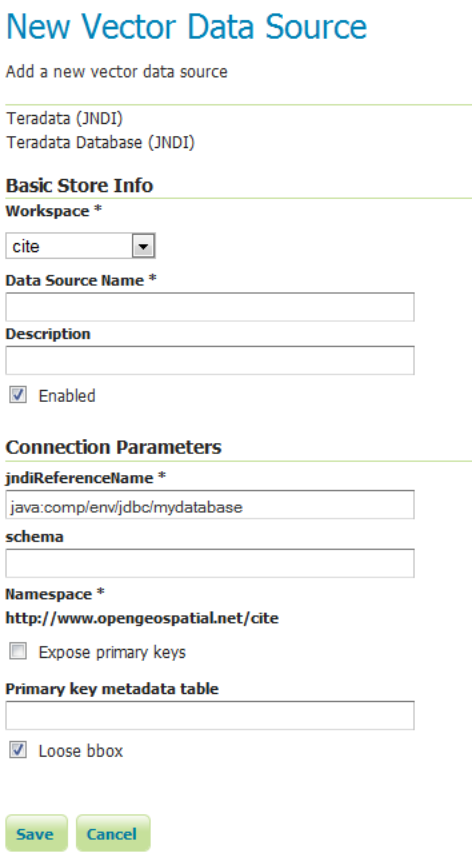


Fig. 5.124: *Adding a Teradata store with JNDI*

Adding layers

One the store has been loaded into GeoServer, the process for loading data layers from database tables is the same as any other database source. Please see the *Layers* section for more information.

Note: Only those database tables that have spatial information and an entry in the `SYSSPATIAL.geometry_columns` table can be served through GeoServer.

GeoServer provides extensive facilities for controlling how databases are accessed. These are covered in the following sections.

5.4.9 Database Connection Pooling

When serving data from a spatial database *connection pooling* is an important aspect of achieving good performance. When GeoServer serves a request that involves loading data from a database table, a connection must first be established with the database. This connection comes with a cost as it takes time to set up such a connection.

The purpose served by a connection pool is to maintain connection to an underlying database between requests. The benefit of which is that connection setup only need to occur once on the first request. Subsequent requests use existing connections and achieve a performance benefit as a result.

Whenever a data store backed by a database is added to GeoServer an internal connection pool is created. This connection pool is configurable.

Connection pool options

max connections	The maximum number of connections the pool can hold. When the maximum number of connections is exceeded, additional requests that require a database connection will be halted until a connection from the pool becomes available. The maximum number of connections limits the number of concurrent requests that can be made against the database.
min connections	The minimum number of connections the pool will hold. This number of connections is held even when there are no active requests. When this number of connections is exceeded due to serving requests additional connections are opened until the pool reaches its maximum size (described above).
validate connections	Flag indicating whether connections from the pool should be validated before they are used. A connection in the pool can become invalid for a number of reasons including network breakdown, database server timeout, etc.. The benefit of setting this flag is that an invalid connection will never be used which can prevent client errors. The downside of setting the flag is that a performance penalty is paid in order to validate connections.
fetch size	The number of records read from the database in each network exchange. If set too low (<50) network latency will affect negatively performance, if set too high it might consume a significant portion of GeoServer memory and push it towards an `Out Of Memory` error. Defaults to 1000.
connection timeout	Time, in seconds, the connection pool will wait before giving up its attempt to get a new connection from the database. Defaults to 20 seconds.
Test while idle	Periodically test if the connections are still valid also while idle in the pool. Sometimes performing a test query using an idle connection can make the datastore unavailable for a while. Often the cause of this troublesome behaviour is related to a network firewall placed between GeoServer and the Database that force the closing of the underlying idle TCP connections.
Evictor run periodicity	Number of seconds between idle object evictor runs.
Max connection idle time	Number of seconds a connection needs to stay idle before the evictor starts to consider closing it.
Evictor tests per run	Number of connections checked by the idle connection evictor for each of its runs.

5.4.10 JNDI

Many data stores and connections in GeoServer have the option of utilizing Java Naming and Directory Interface on JNDI. JNDI allows for components in a Java system to look up other objects and data by a predefined name.

A common use of JNDI is to store a JDBC data source globally in a container. This has a few benefits. First, it can lead to a much more efficient use of database resources. Database connections in Java are very resource-intensive objects, so usually they are pooled. If each component that requires a database connection is responsible for creating their own connection pool, resources will stack up fast. In addition, often those resources are under-utilized and a component may not size its connection pool accordingly. A more efficient method is to set up a global pool at the servlet container level, and have every component that requires a database connection use that.

Furthermore, JNDI consolidates database connection configuration, as not every component that requires a database connection needs to know any more details than the JNDI name. This is very useful for administrators who may have to change database parameters in a running system, as it allows the change to occur in a single place.

5.4.11 SQL Views

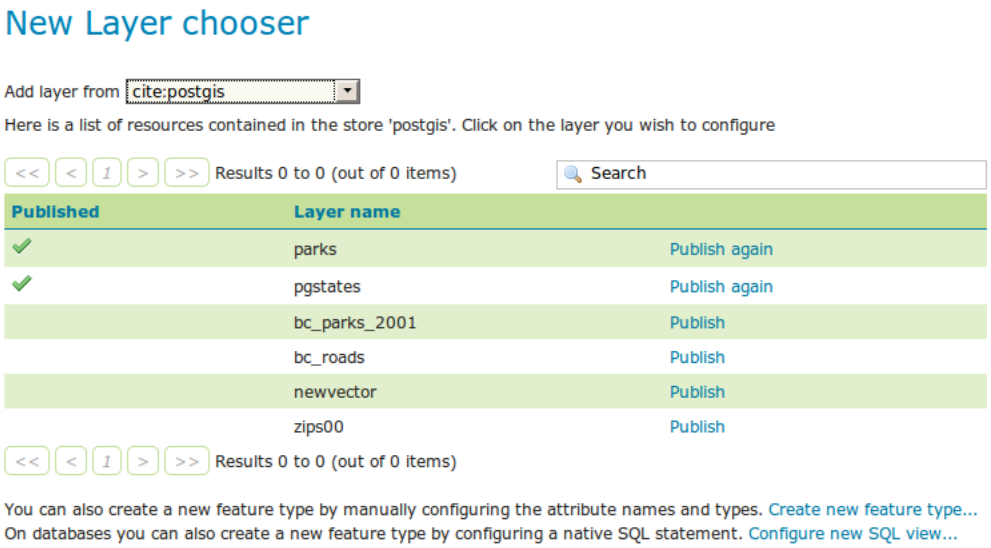
The traditional way to access database data is is to configure layers against either tables or database views. Starting with GeoServer 2.1.0, layers can also be defined as SQL Views. SQL Views allow executing a custom SQL query on each request to the layer. This avoids the need to create a database view for complex queries.

Even more usefully, SQL View queries can be parameterized via string substitution. Parameter values can be supplied in both WMS and WFS requests. Default values can be supplied for parameters, and input values can be validated by Regular Expressions to eliminate the risk of SQL injection attacks.

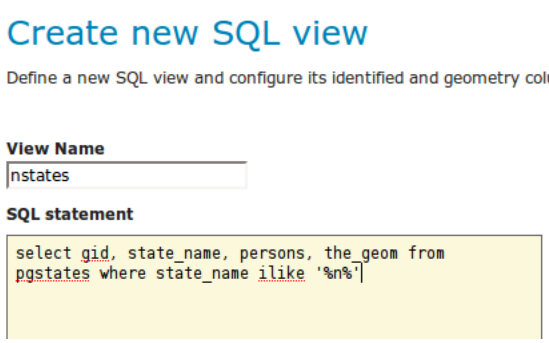
Note: SQL Views are read-only, and thus cannot be updated by WFS-T transactions.

Creating a SQL View

In order to create a SQL View the administrator invokes the *Create new layer* page. When a database store is selected, the usual list of tables and views available for publication appears, A link *Configure new SQL view...* also appears:

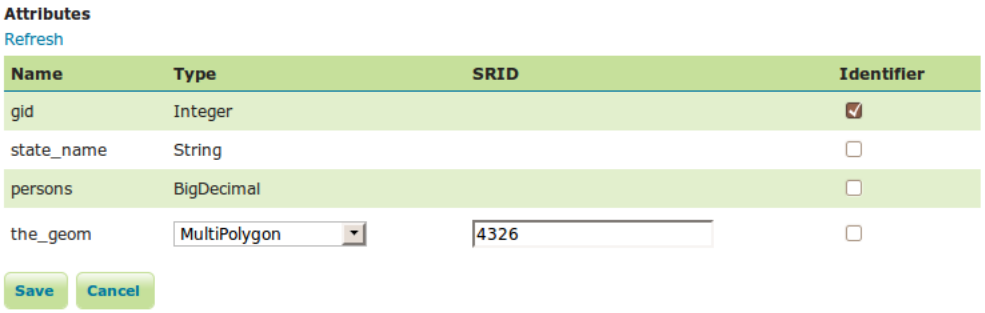


Selecting the *Configure new SQL view...* link opens a new page where the SQL view query can be specified:



Note: The query can be any SQL statement that is valid as a subquery in a FROM clause (that is, `select * from (<the sql view>) [as] vtable`). This is the case for most SQL statements, but in some databases special syntax may be needed to call stored procedures. Also, all the columns returned by the SQL statement must have names. In some databases alias names are required for function calls.

When a valid SQL query has been entered, press the *Refresh* link in the **Attributes** table to get the list of the attribute columns determined from the query:

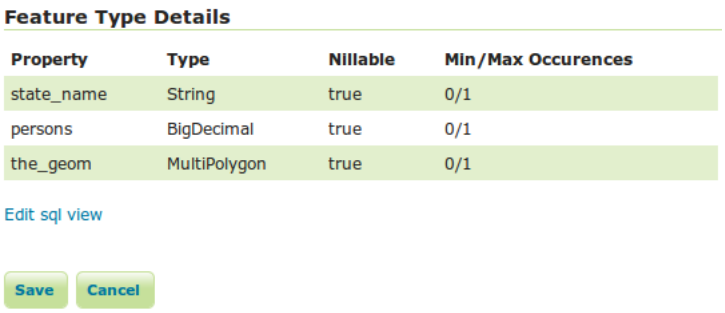


GeoServer attempts to determine the geometry column type and the native SRID, but these should be verified and corrected if necessary.

Note: Having a correct SRID (spatial reference id) is essential for spatial queries to work. In many spatial databases the SRID is equal to the EPSG code for the specific spatial reference system, but this is not always the case (for instance, Oracle has a number of non-EPSG SRID codes).

If stable feature ids are desired for the view's features, one or more columns providing a unique id for the features should be checked in the **Identifier** column. Always ensure these attributes generate a unique key, or filtering and WFS requests will not work correctly.

Once the query and the attribute details are defined, press *Save*. The usual *New Layer* configuration page will appear. If further changes to the view are required, the page has a link to the SQL View editor at the bottom of the *Data* tab:



Once created, the SQL view layer is used in the same way as a conventional table-backed layer, with the one limitation of being read-only.

Warning: Saving the SQL view definition here is not sufficient, the layer containing it must be saved as well for the change to have any effect. This is because the SQL view definition is actually just one component of the layer/featuretype/coverage attributes.

Parameterizing SQL Views

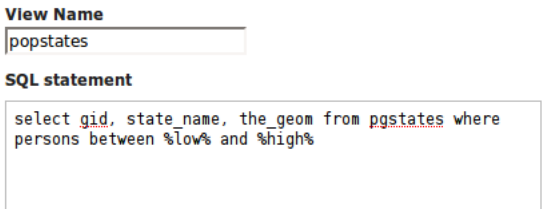
A parametric SQL view is based on a SQL query containing named parameters. The values for the parameters can be provided dynamically in WMS and WFS requests using the `viewparams` request parameter. Parameters can have default values specified, to handle the situation where they are not supplied in a request. Validation of supplied parameter values is supported by specifying validation regular expressions. Parameter values are only accepted if they match the regular expression defined for them. Appropriate parameter validation should always be used to avoid the risk of SQL injection attacks.

Warning: SQL View parameter substitution should be used with caution, since improperly validated parameters open the risk of SQL injection attack. Where possible, consider using safer methods such as *dynamic filtering* in the request, or *Variable substitution in SLD*.

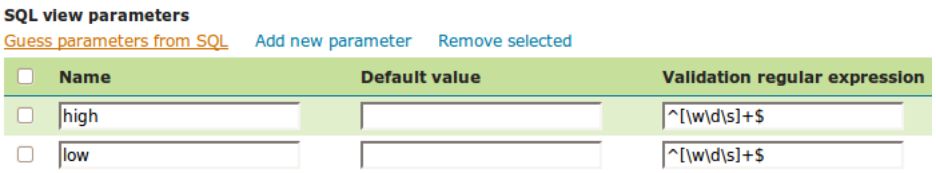
Defining parameters

Within the SQL View query, parameter names are delimited by leading and trailing `%` signs. The parameters can occur anywhere within the query text, including such uses as within SQL string constants, in place of SQL keywords, or representing entire SQL clauses.

Here is an example of a SQL View query for a layer called `popstates` with two parameters, `low` and `high`:

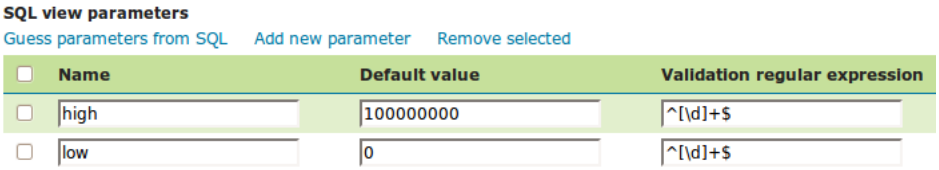


Each parameter needs to be defined with its name, an optional default value, and a validation expression. The *Guess parameters from SQL* link can be clicked to infer the query parameters automatically, or they can be entered manually. The result is a table filled with the parameter names, default values and validation expressions:



In this case the default values should be specified, since the query cannot be executed without values for the parameters (because the expanded query `select gid, state_name, the_geom from pgstates`

`where persons between and` is invalid SQL). Since the use of the parameters in the SQL query requires their values to be positive integer numbers, the validation regular expressions are specified to allow only numeric input (i.e. `^[\d]+$`):



Once the parameters have been defined, the **Attributes** *Refresh* link is clicked to parse the query and retrieve the attribute columns. The computed geometry type and column identifier details can be corrected if required. From this point on the workflow is the same as for a non-parameterized query.

Using a parametric SQL View

The SQL view parameters are specified by adding the `viewparams` parameter to the WMS `GetMap` or the WFS `GetFeature` request. The `viewparams` argument is a list of `key:value` pairs, separated by semicolons:

```
viewparams=p1:v1;p2:v2;...
```

If the values contain semicolons or commas these must be escaped with a backslash (e.g. `\,` and `\;`).

For example, the `popstates` SQL View layer can be displayed by invoking the *Layer Preview*. Initially no parameter values are supplied, so the defaults are used and all the states are displayed,

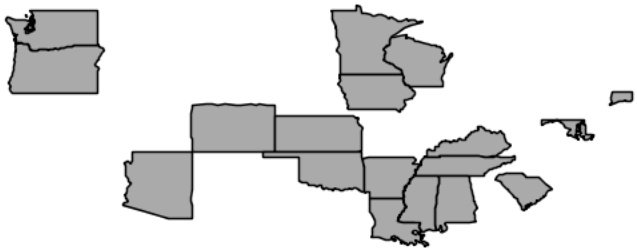
To display all states having more than 20 million inhabitants the following parameter is added to the `GetMap` request: `&viewparams=low:20000000`

To display all states having between 2 and 5 millions inhabitants the view parameters are: `&viewparams=low:2000000;high:5000000`

Parameters can be provided for multiple layers by separating each parameter map with a comma:

```
&viewparams=l1p1:v1;l1p2:v2,l2p1:v1;l2p2:v2,...
```

The number of parameter maps must match the number of layers (featuretypes) included in the request.



Parameters and validation

The value of a SQL View parameter can be an arbitrary string of text. The only constraint is that the attribute names and types returned by the view query must never change. This makes it possible to create views containing parameters representing complex SQL fragments. For example, using the view query `select * from pgstates %where%` allows specifying the WHERE clause of the query dynamically. However, this would likely require an empty validation expression. which presents a serious risk of SQL injection attacks. This technique should only be used if access to the server is restricted to trusted clients.

In general, SQL parameters must be used with care. They should always include validation regular expressions that accept only the intended parameter values. Note that while validation expressions should be constructed to prevent illegal values, they do not necessarily have to ensure the values are syntactically correct, since this will be checked by the database SQL parser. For example:

- `^[\d\.\+-eE]+$` checks that a parameter value contains valid characters for floating-point numbers (including scientific notation), but does not check that the value is actually a valid number
- `[^;']+` checks that a parameter value does not contain quotes or semicolons. This prevents common SQL injection attacks, but otherwise does not impose much limitation on the actual value

Resources for Validation Regular expressions

Defining effective validation regular expressions is important for security. Regular expressions are a complex topic that cannot be fully addressed here. The following are some resources for constructing regular expressions:

- GeoServer uses the standard Java regular expression engine. The Pattern class Javadocs contain the full specification of the allowed syntax.
- http://www.regular-expressions.info has many tutorials and examples of regular expressions.
- The myregexp applet can be used to test regular expressions online.

Place holder for the SQL WHERE clause

The SQL `WHERE` clause produced by GeoServer using the context filters, e.g. the bounding box filter of a WMS query, will be added around the SQL view definition. This comes handy (better performance) when we have extra operations that can be done on top of the rows filtered with the GeoServer produced filter first.

A typical use case for this functionality is the execution of analytic functions on top of the filtered results:

```
SELECT STATION_NAME,
       MEASUREMENT,
       MEASUREMENT_TYPE,
       LOCATION
FROM
  (SELECT STATION_NAME,
          MEASUREMENT,
          MEASUREMENT_TYPE,
          LOCATION,
          ROW_NUMBER() OVER(PARTITION BY STATION_ID, MEASUREMENT_TYPE
                            ORDER BY TIME DESC) AS RANK
   FROM
     (SELECT st.id AS STATION_ID,
             st.common_name AS STATION_NAME,
             ob.value AS MEASUREMENT,
             pr.param_name AS MEASUREMENT_TYPE,
             ob.time AS TIME,
             st.position AS LOCATION
      FROM meteo.meteo_stations st
      LEFT JOIN meteo.meteo_observations ob ON st.id = ob.station_id
      LEFT JOIN meteo.meteo_parameters pr ON ob.parameter_id = pr.id

      -- SQL WHERE clause place holder for GeoServer
      WHERE 1 = 1 :where_clause:) AS stations_filtered) AS stations

WHERE RANK = 1;
```

A few restrictions apply when using the explicit `:where_clause:` place holder:

- it needs to be added in a position where all the attributes known by GeoServer are already present
- the `:where_clause:` can only appear once

When a `WHERE` clause place holder is present, GeoServer will always add an explicit `AND` at the beginning of the produced `WHERE` clause. This allows the injection of the produced `WHERE` in the middle of complex expressions if needed.

5.4.12 Controlling feature ID generation in spatial databases

Introduction

All spatial database data stores (PostGIS, Oracle, MySQL and so on) normally derive the feature ID from the table primary key and assume certain conventions on how to locate the next value for the key in case a new feature needs to be generated (WFS insert operation).

Common conventions rely on finding auto-increment columns (PostGIS) or finding a sequence that is named after a specific convention such as `<table>_<column>_SEQUENCE` (Oracle case).

In case none of the above is found, normally the store will fall back on generating random feature IDs at each new request, making the table unsuitable for feature ID based searches and transactions.

Metadata table description

These defaults can be overridden manually by creating a *primary key metadata table* that specifies which columns to use and what strategy to use to generate new primary key values. The (schema qualified) table

can be created with this SQL statement (this one is valid for PostGIS and ORACLE, adapt it to your specific database, you may remove the check at the end if you want to):

```
--PostGIS DDL

CREATE TABLE my_schema.gt_pk_metadata (
  table_schema VARCHAR(32) NOT NULL,
  table_name VARCHAR(32) NOT NULL,
  pk_column VARCHAR(32) NOT NULL,
  pk_column_idx INTEGER,
  pk_policy VARCHAR(32),
  pk_sequence VARCHAR(64),
  unique (table_schema, table_name, pk_column),
  check (pk_policy in ('sequence', 'assigned', 'autogenerated'))
)


--ORACLE DDL

CREATE TABLE gt_pk_metadata (
  table_schema VARCHAR2(32) NOT NULL,
  table_name VARCHAR2(32) NOT NULL,
  pk_column VARCHAR2(32) NOT NULL,
  pk_column_idx NUMBER(38),
  pk_policy VARCHAR2(32),
  pk_sequence VARCHAR2(64),
  constraint  chk_pk_policy check (pk_policy in ('sequence', 'assigned',
↪'autogenerated')));

CREATE UNIQUE INDEX gt_pk_metadata_table_idx01 ON gt_pk_metadata (table_schema, table_
↪name, pk_column);
```

The table can be given a different name. In that case, the (schema qualified) name of the table must be specified in the *primary key metadata table* configuration parameter of the store.

The following table describes the meaning of each column in the metadata table.

Column	Description
table_schema	Name of the database schema in which the table is located.
table_name	Name of the table to be published
pk_column	Name of a column used to form the feature IDs
pk_column_idx	Index of the column in a multi-column key. In case multi column keys are needed multiple records with the same table schema and table name will be used.
pk_policy	The new value generation policy, used in case a new feature needs to be added in the table (following a WFS-T insert operation).
pk_sequence	The name of the database sequence to be used when generating a new value for the pk_column.

The possible values are:

- *assigned*. The value of the attribute in the newly inserted feature will be used (this assumes the "expose primary keys" flag has been enabled)
- *sequence*. The value of the attribute will be generated from the next value of a sequence indicated in the "pk_sequence" column.
- *autogenerated*. The column is an auto-increment one, the next value in the auto-increment will be used.

Using the metadata table with views

GeoServer can publish spatial views without issues, but normally results in two side effects:

- the view is treated as read only
- the feature IDs are randomly generated

The metadata table can also refer to views, just use the view name in the `table_name` column: this will result in stable ids, and in databases supporting updatable views, it will also make the code treat the view as writable (thus, enabling usage of WFS-T on it).

5.4.13 Custom SQL session start/stop scripts

Starting with version 2.1.4 GeoServer support custom SQL scripts that can be run every time GeoServer grabs a connection from the connection pool, and every time the session is returned to the pool.

These scripts can be parametrized with the expansion of environment variables, which can be in turn set into the OGC request parameters with the same mechanism as *Variable substitution in SLD*.

In addition to the parameters provided via the request the `GSUSER` variable is guaranteed to contain the current GeoServer user, or be null if no authentication is available. This is useful if the SQL sessions scripts are used to provide tight control over database access

The SQL script can expand environment variables using the `${variableName, defaultValue}` syntax, for example the following alters the current database user to be the same as the GeoServer current user, or `geoserver` in case no user was authenticated

```
SET SESSION AUTHORIZATION ${GSUSER,geoserver}
```

5.4.14 Using SQL session scripts to control authorizations at the database level

GeoServer connects to a database via a connection pool, using the same rights as the user that is specified in the connection pool setup. In a setup that provides a variety of services and tables the connection pool user must have a rather large set of rights, such as table selection (WMS), table insert/update/delete (WFS-T) and even table creation (data upload via RESTConfig, WPS Import process and eventual new processes leveraging direct database connections).

What a user can do can be controlled by means of the GeoServer security subsystem, but in high security setups this might not be considered enough, and a database level access control be preferred instead. In these setups normally the connection pool user has limited access, such as simple read only access, while specific users are allowed to perform more operations.

When setting up such a solution remember the following guidelines:

- The connection pool user must be able to access all table metadata regardless of whether it is able to actually perform a select on the tables (dictionary tables/describe functionality must be always accessible)
- The connection pool must see each and every column of tables and views, in other words, the structure of the tables must not change as the current user changes
- the database users and the GeoServer user must be kept in synch with some external tools, GeoServer provides no out of the box facilities
- during the GeoServer startup the code will access the database to perform some sanity checks, in that moment there is no user authenticated in GeoServer so the code will run under whatever user was specified as the "default value" for the `GSUSER` variable.

- The user that administers GeoServer (normally `admin`, but it can be renamed, and other users given the administration roles too) must also be a database user, all administrative access on the GeoServer GUI will have that specific user controlling the session

Typical use cases:

- Give insert/update/delete rights only to users that must use WFS-T
- Only allow the administrator to create new tables
- Limit what rows of a table a user can see by using dynamic SQL views taking into account the current user to decide what rows to return

To make a point in case, if we want the PostgreSQL session to run with the current GeoServer user credentials the following scripts will be used:

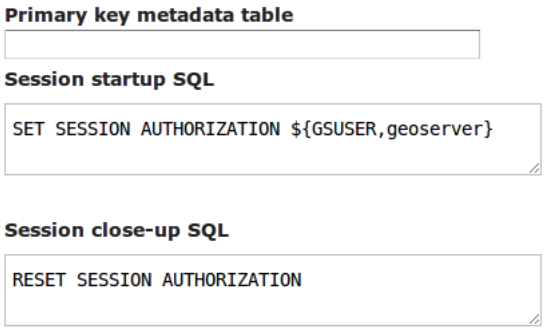


Fig. 5.125: *Setting up session authorization for PostgreSQL*

The first command makes the database session use either the current GeoServer user, or the `geoserver` user if no authentication was available (anonymous user, or startup situation). The second command resets the session to the rights of the connection pool user.

5.5 Cascaded service data

This section discusses how GeoServer can proxy external OGC services. This is known as **cascading** services.

GeoServer supports cascading the following services:

5.5.1 External Web Feature Server

GeoServer has the ability to load data from a remote Web Feature Server (WFS). This is useful if the remote WFS lacks certain functionality that GeoServer contains. For example, if the remote WFS is not also a Web Map Server (WMS), data from the WFS can be cascaded through GeoServer to utilize GeoServer's WMS. If the remote WFS has a WMS but that WMS cannot output KML, data can be cascaded through GeoServer's WMS to output KML.

Adding an external WFS

To connect to an external WFS, it is necessary to load it as a new datastore. To start, navigate to *Stores* → *Add a new store* → *Web Feature Server*.

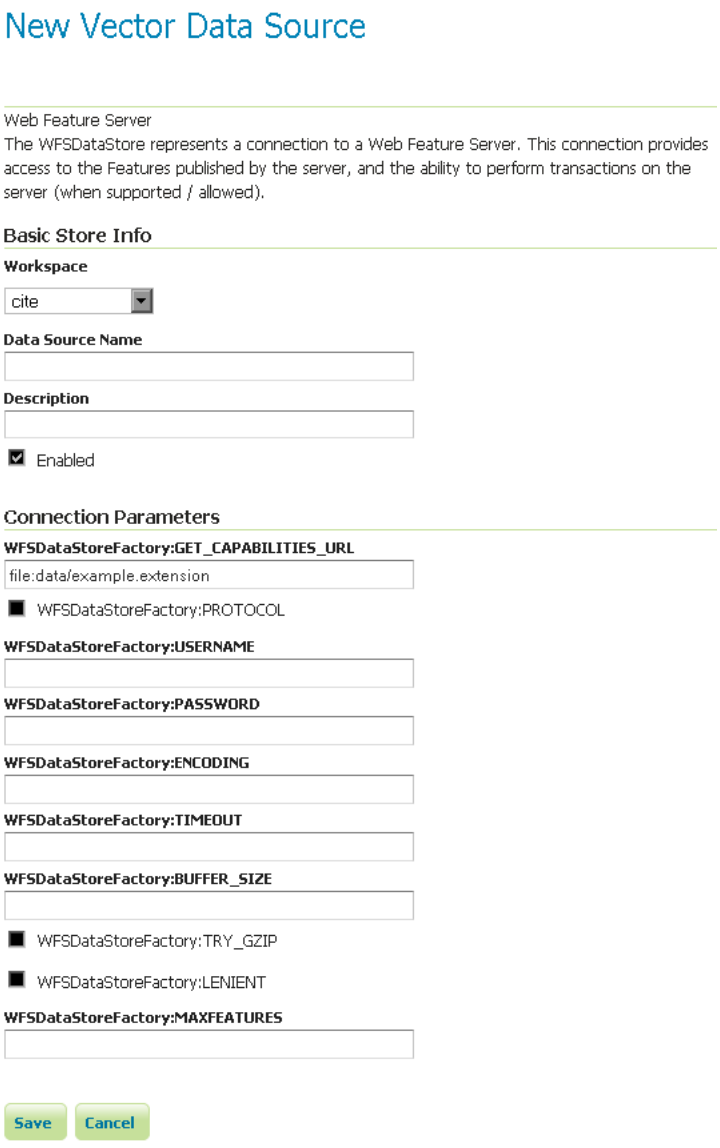


Fig. 5.126: *Adding an external WFS as a store*

Option	Description
Workspace	Name of the workspace to contain the store. This will also be the prefix of all of the layer names created from the store.
Data Source Name	Name of the store as known to GeoServer.
Description	Description of the store.
Enabled	Enables the store. If disabled, no data from the external WFS will be served.
GET_CAPABILITIES_URL	URL to access the capabilities document of the remote WFS.
PROTOCOL	When checked, connects with POST, otherwise uses GET.
USERNAME	The user name to connect to the external WFS.
PASSWORD	The password associated with the above user name.
ENCODING	The character encoding of the XML requests sent to the server. Defaults to `UTF-8`.
TIMEOUT	Time (in milliseconds) before timing out. Default is `3000`.
BUFFER_SIZE	Specifies a buffer size (in number of features). Default is `10` features.
TRY_GZIP	Specifies that the server should transfer data using compressed HTTP if supported by the server.
LENIENT	When checked, will try to render features that don't match the appropriate schema. Errors will be logged.
MAXFEATURES	Maximum amount of features to retrieve for each featuretype. Default is no limit.
AXIS_ORDER	Axis order used in result coordinates (It applies only to WFS 1.x.0 servers). Default is Compliant.
AXIS_ORDER_FILTER	Axis order used in filter (It applies only to WFS 1.x.0 servers). Default is Compliant.
OUTPUTFORMAT	Output format to request (instead of the default remote service one).
GML_COMPLIANCE_LEVEL	GML compliance level. i.e. (simple feature) 0, 1 or 2. Default is 0.
GML_COMPATIBLE_TYPENAMES	Use GML compatible TypeNames (replace : by _). Default is no false.
USE_HTTP_CONNECTION_POOLING	Use connection pooling to connect to the remote WFS service. Also enables digest authentcation.

When finished, click *Save*.

Configuring external WFS layers

When properly loaded, all layers served by the external WFS will be available to GeoServer. Before they can be served, however, they will need to be individually configured as new layers. See the section on *Layers* for how to add and edit new layers.

Connecting to an external WFS layer via a proxy server

In a corporate environment it may be necessary to connect to an external WFS through a proxy server. To achieve this, various java variables need to be set.

For a Windows install running GeoServer as a service, this is done by modifying the wrapper.conf file. For a default Windows install, modify `C:\Program Files\GeoServer x.x.x\wrapper\wrapper.conf` similarly to the following.

```
# Java Additional Parameters

wrapper.java.additional.1=-Djetty.home=. wrapper.java.additional.2=-
DGEOSERVER_DATA_DIR="%GEOSERVER_DATA_DIR%" wrapper.java.additional.3=-
Dhttp.proxySet=true wrapper.java.additional.4=-Dhttp.proxyHost=maitproxy
wrapper.java.additional.5=-Dhttp.proxyPort=8080 wrapper.java.additional.6=-
Dhttps.proxyHost=maitproxy wrapper.java.additional.7=-Dhttps.proxyPort=8080
wrapper.java.additional.8=-Dhttp.nonProxyHosts="mait*|dpi*|localhost"
```

Note that the **`http.proxySet=true`** parameter is required. Also, the parameter numbers must be consecutive - ie. no gaps.

For a Windows install not running GeoServer as a service, modify `startup.bat` so that the **`java`** command runs with similar -D parameters.

For a Linux/UNIX install, modify `startup.sh` so that the **`java`** command runs with similar -D parameters.

5.5.2 Cascaded Web Feature Service Stored Queries

Stored query is a feature of Web Feature Services. It allows servers to serve pre-configured filter queries or even queries that cannot be expressed as GetFeature filter queries. This feature of GeoServer allows to create cascaded layers out of stored queries.

Stored queries are read-only, and layers derived from cascaded stored queries cannot be updated with WFS-T.

Cascaded stored query parameters

The relationship between stored query parameters and the schema returned by the query is not well defined. For cascaded stored queries to work, the relationship between the query received by GeoServer and the parameters passed to the stored query must be defined.

When you set up a layer based on a stored query, you have to select which stored query to cascade and what values are passed to each parameter. Cascaded stored queries can leverage view parameters passed to the query. This is similar to how arbitrary parameters are passed to *SQL Views*. GeoServer supports multiple strategies to pass these values. See below for a full list.

Parameter type	Explanation
No mapping	The value of the view parameter will be passed as is to the stored query. No parameter will be passed if there is no view parameter of the same name.
Blocked	This parameter will never be passed to the stored query
Default	The specified value is used unless overwritten by a view parameter
Static	The specified value is always used (view parameter value will be ignored)
CQL Expression	An expression that will be evaluated on every request (see below for more details)

See *Using a parametric SQL View* for more details how clients pass view parameters to GeoServer.

CQL expressions

Parameter mappings configured as CQL expressions are evaluated for each request using a context derived from the request query and the view parameters. General information on CQL expressions is available here *Expression*.

The context contains the following properties that may be used in the expressions:

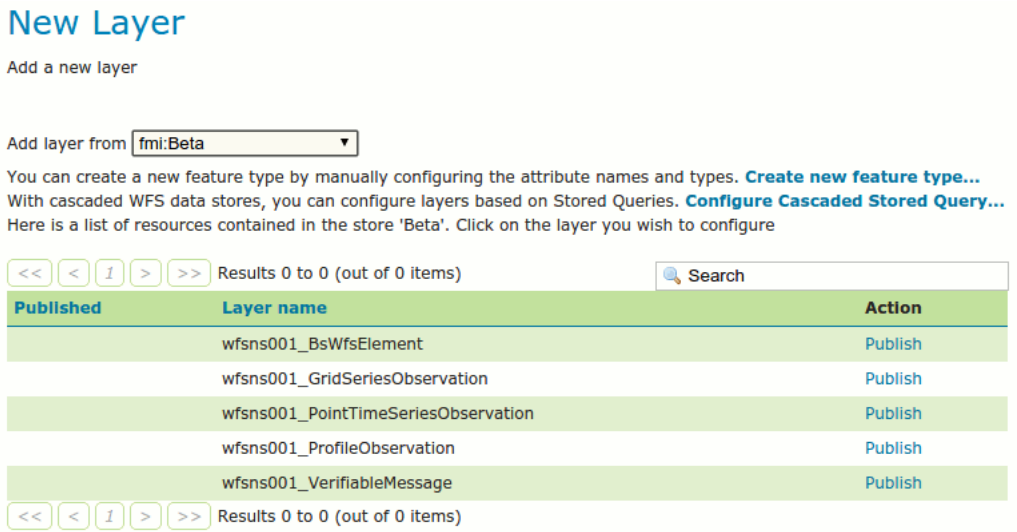
Property name	Explanation

`bboxMinX`
`bboxMinY`
`bboxMaxX`
`bboxMaxY`
	Evaluates to a corner coordinate of the full extent of the query

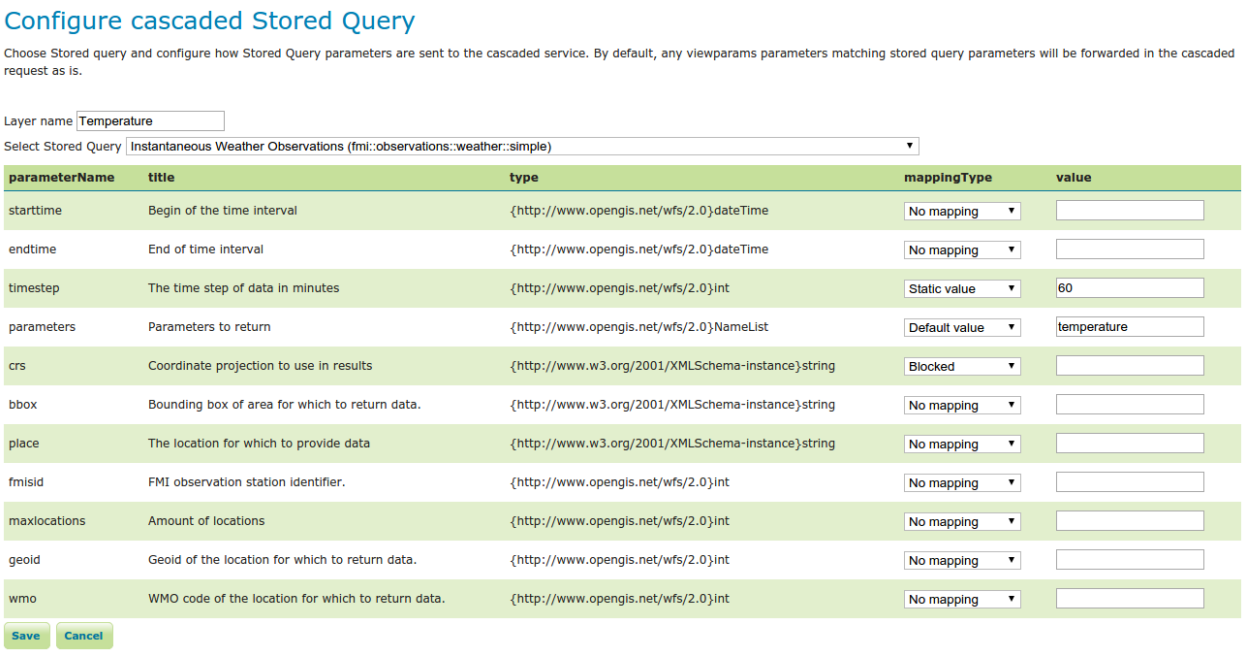
`defaultSRS`	Evaluates to the default SRS of the feature type
`viewparam:name`	Evaluates to the value of the view parameter *name* in this query

Configuring a cascaded stored query layer

In order to create a cascaded stored query layer the administrator invokes the Create new layer page. When an *External Web Feature Server* is selected, the usual list of tables and views available for publication appears, a link *Configure Cascaded Stored Query...* also appears:



Selecting the *Configure Cascaded Stored Query...* link opens a new page where the parameter mapping is set up. By default all parameters are set up as *No mapping*.



5.5.3 External Web Map Server

GeoServer has the ability to proxy a remote Web Map Service (WMS). This process is sometimes known as **Cascading WMS**. Loading a remote WMS is useful for many reasons. If you don't manage or have access

to the remote WMS, you can now manage its output as if it were local. Even if the remote WMS is not GeoServer, you can use GeoServer features to treat its output (watermarking, decoration, printing, etc).

To access a remote WMS, it is necessary to load it as a store in GeoServer. GeoServer must be able to access the capabilities document of the remote WMS for the store to be successfully loaded.

Adding an external WMS

To connect to an external WMS, it is necessary to load it as a new store. To start, in the *Web administration interface*, navigate to *Stores* → *Add a new store* → *WMS*. The option is listed under *Other Data Sources*.

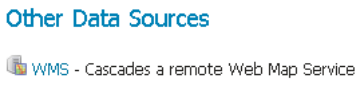


Fig. 5.127: *Adding an external WMS as a store*

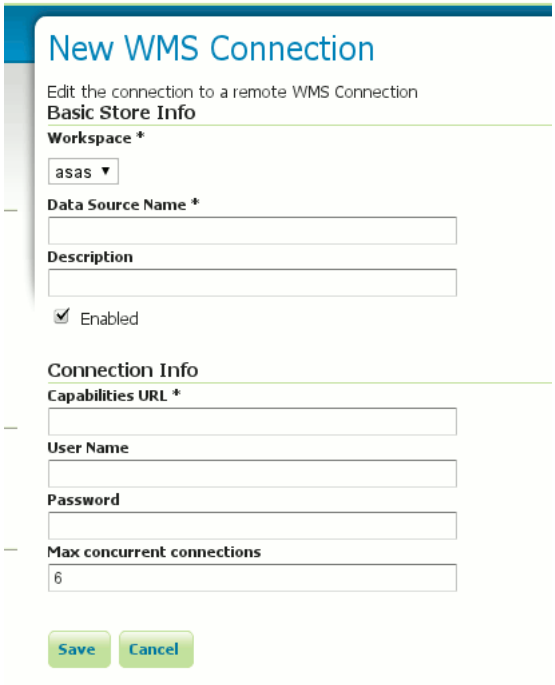


Fig. 5.128: *Configuring a new external WMS store*

Option	Description
Workspace	Name of the workspace to contain the store. This will also be the prefix of all of the layer names published from the store. **The workspace name on the remote WMS is not cascaded.**
Data Source Name	Name of the store as known to GeoServer.
Description	Description of the store.
Enabled	Enables the store. If disabled, no data from the remote WMS will be served.
Capabilities URL	The full URL to access the capabilities document of the remote WMS.
User Name	If the WMS requires authentication, the user name to connect as.
Password	If the WMS requires authentication, the password to connect with.
Max concurrent connections	The maximum number of persistent connections to keep for this WMS.

When finished, click *Save*.

Configuring external WMS layers

When properly loaded, all layers served by the external WMS will be available to GeoServer. Before they can be served, however, they will need to be individually configured (published) as new layers. See the section on *Layers* for how to add and edit new layers. Once published, these layers will show up in the *Layer Preview* and as part of the WMS capabilities document.

Features

Connecting a remote WMS allows for the following features:

- **Dynamic reprojection**. While the default projection for a layer is cascaded, it is possible to pass the SRS parameter through to the remote WMS. Should that SRS not be valid on the remote server, GeoServer will dynamically reproject the images sent to it from the remote WMS.
- **GetFeatureInfo**. WMS GetFeatureInfo requests will be passed to the remote WMS. If the remote WMS supports the `application/vnd.ogc.gml` format the request will be successful.
- Full **REST Configuration**. See the *REST* section for more information about the GeoServer REST interface.

Limitations

Layers served through an external WMS have some, but not all of the functionality of a local WMS.

- Layers cannot be styled with SLD.
- Alternate (local) styles cannot be used.
- Extra request parameters (`time`, `elevation`, `cql_filter`, etc.) cannot be used.
- GetLegendGraphic requests aren't supported.
- Image format cannot be specified. GeoServer will attempt to request PNG images, and if that fails will use the remote server's default image format.

5.5.4 External Web Map Tile Server

GeoServer has the ability to proxy a remote Web Map Tile Service (WMTS). This process is sometimes known as **Cascading WMTS**, even if the incoming requests follow the WMS protocol and the backing service follows the WMTS one; the WMTS cascading functionality is more like a "protocol translator", where the different handled data (capabilities documents, images) are translated by the "WMTS cascading" logic.

Loading a remote WMTS is useful for many reasons. If you don't manage or have access to the remote WMTS, you can now manage its output as if it were local. Even if you don't have any control on the remote WMTS, you can use GeoServer features to treat its output (watermarking, decoration, printing, etc).

To access a remote WMTS, it is necessary to load it as a store in GeoServer. GeoServer must be able to access the capabilities document of the remote WMTS for the store to be successfully loaded.

Adding an external WMTS

To connect to an external WMTS, it is necessary to load it as a new store. To start, in the *Web administration interface*, navigate to *Stores* → *Add a new store* → *WMTS*. The option is listed under *Other Data Sources*.

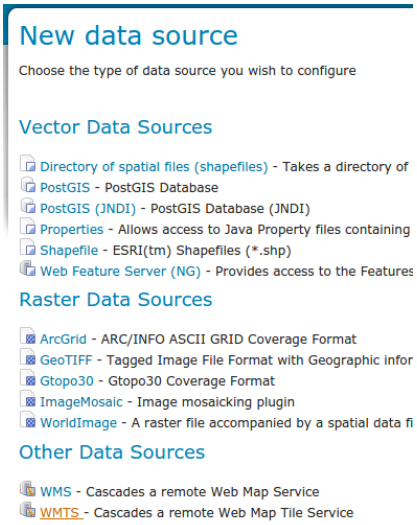


Fig. 5.129: *Adding an external WMTS as a store*

Option	Description
Workspace	Name of the workspace to contain the store. This will also be the prefix of all of the layer names published from the store.
Data Source Name	Name of the store as known to GeoServer.
Description	Description of the store.
Enabled	Enables the store. If disabled, no data from the remote WMTS will be served.
Capabilities URL	The full URL to access the capabilities document of the remote WMTS.
User Name	If the WMTS requires authentication, the user name to connect as.
Password	If the WMTS requires authentication, the password to connect with.
HTTP header name	If the WMTS requires a custom HTTP header, the header name.
HTTP header value	If the WMTS requires a custom HTTP header, the header value.
Max concurrent connections	The maximum number of persistent connections to keep for this WMTS.

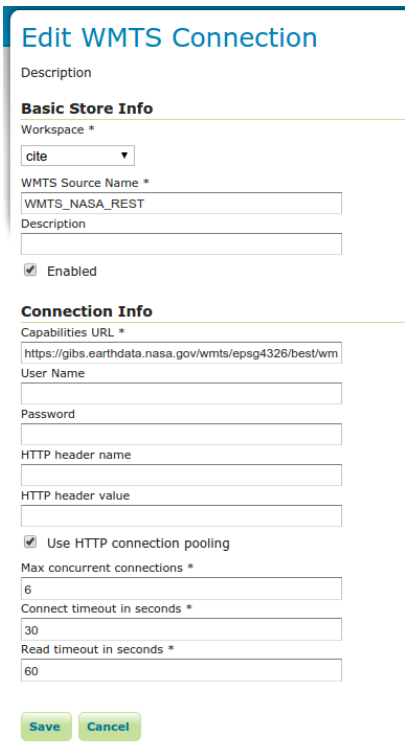


Fig. 5.130: *Configuring a new external WTMS store*

When finished, click *Save*.

Configuring external WMTS layers

When properly loaded, all layers served by the external WMTS will be available to GeoServer. Before they can be served, however, they will need to be individually configured (published) as new layers. See the section on *Layers* for how to add and edit new layers. Once published, these layers will show up in the *Layer Preview* and as part of the WMS capabilities document. If the WMTS layer has additional dimensions (e.g. time), related info will be reported on the WMS capabilities as well.

Features

Connecting a remote WMTS allows for the following features:

- **Dynamic reprojection**. While the default projection for a layer is cascaded, it is possible to pass the SRS parameter through to the remote WMS. Should that SRS not be valid on the remote server, GeoServer will dynamically reproject the tiles sent to it from the remote WMTS.
- Full **REST Configuration**. See the *REST* section for more information about the GeoServer REST interface.

Limitations

Layers served through an external WMTS have some, but not all of the functionality of a local layer.

- Layers cannot be styled with SLD.

- Alternate (local) styles cannot be used.
- GetFeatureInfo requests aren't supported.
- GetLegendGraphic requests aren't supported.
- Image format cannot be specified. GeoServer will attempt to request PNG images, and if that fails will use the remote server's default image format.

5.6 Application schemas

The application schema support (app-schema) extension provides support for *Complex Features* in GeoServer WFS.

Note: You must install the app-schema plugin to use Application Schema Support.

GeoServer provides support for a broad selection of simple feature data stores, including property files, shapefiles, and JDBC data stores such as PostGIS and Oracle Spatial. The app-schema module takes one or more of these simple feature data stores and applies a mapping to convert the simple feature types into one or more complex feature types conforming to a GML application schema.

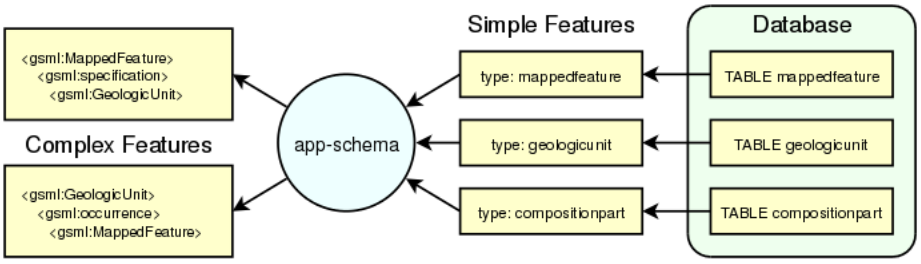


Fig. 5.131: *Three tables in a database are accessed using GeoServer simple feature support and converted into two complex feature types.*

The app-schema module looks to GeoServer just like any other data store and so can be loaded and used to service WFS requests. In effect, the app-schema data store is a wrapper or adapter that converts a simple feature data store into complex features for delivery via WFS. The mapping works both ways, so queries against properties of complex features are supported.

5.6.1 Complex Features

To understand complex features, and why you would want use them, you first need to know a little about simple features.

Simple features

A common use of GeoServer WFS is to connect to a data source such as a database and access one or more tables, where each table is treated as a WFS simple feature type. Simple features contain a list of properties that each have one piece of simple information such as a string or number. (Special provision is made for geometry objects, which are treated like single items of simple data.) The Open Geospatial Consortium

(OGC) defines three Simple Feature profiles; SF-0, SF-1, and SF-2. GeoServer simple features are close to OGC SF-0, the simplest OGC profile.

GeoServer WFS simple features provide a straightforward mapping from a database table or similar structure to a "flat" XML representation, where every column of the table maps to an XML element that usually contains no further structure. One reason why GeoServer WFS is so easy to use with simple features is that the conversion from columns in a database table to XML elements is automatic. The name of each element is the name of the column, in the namespace of the data store. The name of the feature type defaults to the name of the table. GeoServer WFS can manufacture an XSD type definition for every simple feature type it serves. Submit a DescribeFeatureType request to see it.

Benefits of simple features

- Easy to implement
- Fast
- Support queries on properties, including spatial queries on geometries

Drawbacks of simple features

- When GeoServer automatically generates an XSD, the XML format is tied to the database schema.
- To share data with GeoServer simple features, participants must either use the same database schema or translate between different schemas.
- Even if a community could agree on a single database schema, as more data owners with different data are added to a community, the number of columns in the table becomes unmanageable.
- Interoperability is difficult because simple features do not allow modification of only part of the schema.

Simple feature example

For example, if we had a database table `stations` containing information about GPS stations:

```
| id | code |      name      |          location          |
+----+------+----------------+----------------------------+
| 27 | ALIC | Alice Springs  | POINT(133.8855 -23.6701) |
| 4  | NORF | Norfolk Island | POINT(167.9388 -29.0434) |
| 12 | COCO | Cocos          | POINT(96.8339 -12.1883)  |
| 31 | ALBY | Albany         | POINT(117.8102 -34.9502) |
```

GeoServer would then be able to create the following simple feature WFS response fragment:

```
<gps:stations gml:id="stations.27">
    <gps:code>ALIC</gps:code>
    <gps:name>Alice Springs</gps:name>
    <gps:location>
        <gml:Point srsName="urn:x-ogc:def:crs:EPSG:4326">
            <gml:pos>-23.6701 133.8855</gml:pos>
        </gml:Point>
    </gps:location>
</gps:stations>
```

- Every row in the table is converted into a feature.

- Every column in the table is converted into an element, which contains the value for that row.
- Every element is in the namespace of the data store.
- Automatic conversions are applied to some special types like geometries, which have internal structure, and include elements defined in GML.

Complex features

Complex features contain properties that can contain further nested properties to arbitrary depth. In particular, complex features can contain properties that are other complex features. Complex features can be used to represent information not as an XML view of a single table, but as a collection of related objects of different types.

Simple feature	Complex feature
Properties are single data item, e.g. text, number, geometry	Properties can be complex, including complex features
XML view of a single table	Collection of related identifiable objects
Schema automatically generated based on database	Schema agreed by community
One large type	Multiple different types
Straightforward	Richly featured data standards
Interoperability relies on simplicity and customisation	Interoperability through standardization

Benefits of complex features

- Can define information model as an object-oriented structure, an *application schema*.
- Information is modelled not as a single table but as a collection of related objects whose associations and types may vary from feature to feature (polymorphism), permitting rich expression of content.
- By breaking the schema into a collection of independent types, communities need only extend those types they need to modify. This simplifies governance and permits interoperability between related communities who can agree on common base types but need not agree on application-specific subtypes..

Drawbacks of complex features

- More complex to implement
- Complex responses might slower if more database queries are required for each feature.
- Information modelling is required to standardize an application schema. While this is beneficial, it requires effort from the user community.

Complex feature example

Let us return to our `stations` table and supplement it with a foreign key `gu_id` that describes the relationship between the GPS station and the geologic unit to which it is physically attached:

```
| id | code |      name      |          location          | gu_id |
+----+------+----------------+----------------------------+-------+
| 27 | ALIC | Alice Springs  | POINT(133.8855 -23.6701) | 32785 |
| 4  | NORF | Norfolk Island | POINT(167.9388 -29.0434) | 10237 |
| 12 | COCO | Cocos          | POINT(96.8339 -12.1883)  | 19286 |
| 31 | ALBY | Albany         | POINT(117.8102 -34.9502) | 92774 |
```

The geologic unit is is stored in the table `geologicunit`:

```
| gu_id |                       urn              |         text        |
+-------+----------------------------------------+---------------------+
| 32785 | urn:x-demo:feature:GeologicUnit:32785 | Metamorphic bedrock |
...
```

The simple features approach would be to join the `stations` table with the `geologicunit` table into one view and then deliver "flat" XML that contained all the properties of both. The complex feature approach is to deliver the two tables as separate feature types. This allows the relationship between the entities to be represented while preserving their individual identity.

For example, we could map the GPS station to a `sa:SamplingPoint` with a `gsml:GeologicUnit`. The these types are defined in the following application schemas respectively:

- http://schemas.opengis.net/sampling/1.0.0/sampling.xsd
  - Documentation: OGC 07-002r3: http://portal.opengeospatial.org/files/?artifact_id=22467
- http://www.geosciml.org/geosciml/2.0/xsd/geosciml.xsd
  - Documentation: http://www.geosciml.org/geosciml/2.0/doc/

The complex feature WFS response fragment could then be encoded as:

```
<sa:SamplingPoint gml:id="stations.27>
  <gml:name codeSpace="urn:x-demo:SimpleName">Alice Springs</gml:name>
  <gml:name codeSpace="urn:x-demo:IGS:ID">ALIC</gml:name>
  <sa:sampledFeature>
     <gsml:GeologicUnit gml:id="geologicunit.32785">
         <gml:description>Metamorphic bedrock</gml:description>
         <gml:name codeSpace="urn:x-demo:Feature">urn:x-
↪demo:feature:GeologicUnit:32785</gml:name>
     </gsml:GeologicUnit>
  </sa:sampledFeature>
  <sa:relatedObservation xlink:href="urn:x-demo:feature:GeologicUnit:32785" />
  <sa:position>
      <gml:Point srsName="urn:x-ogc:def:crs:EPSG:4326">
          <gml:pos>-23.6701 133.8855</gml:pos>
      </gml:Point>
  </sa:position>
</sa:SamplingPoint>
```

- The property `sa:sampledFeature` can reference any other feature type, inline (included in the response) or by reference (an `xlink:href` URL or URN). This is an example of the use of polymorphism.
- The property `sa:relatedObservation` refers to the same GeologicUnit as `sa:sampledFeature`, but by reference.
- Derivation of new types provides an extension point, allowing information models to be reused and extended in a way that supports backwards compatibility.

- Multiple sampling points can share a single GeologicUnit. Application schemas can also define multivalued properties to support many-to-one or many-to-many associations.
- Each GeologicUnit could have further properties describing in detail the properties of the rock, such as colour, weathering, lithology, or relevant geologic events.
- The GeologicUnit feature type can be served separately, and could be uniquely identified through its properties as the same instance seen in the SamplingPoint.

Portrayal complex features (SF0)

Portrayal schemas are standardized schemas with flat attributes, also known as simple feature level 0 (SF0). Because a community schema is still required (e.g. GeoSciML-Portrayal), app-schema plugin is still used to map the database columns to the attributes.

- *WFS CSV output format* is supported for complex features with portrayal schemas. At the moment, propertyName selection is not yet supported with csv outputFormat, so it always returns the full set of attributes.
- Complex features with nesting and multi-valued properties are not supported with *WFS CSV output format*.

5.6.2 Installation

Application schema support is a GeoServer extension and is downloaded separately.

- Download the app-schema plugin zip file for the same version of GeoServer.
- Unzip the app-schema plugin zip file to obtain the jar files inside. Do not unzip the jar files.
- Place the jar files in the `WEB-INF/lib` directory of your GeoServer installation.
- Restart GeoServer to load the extension (although you might want to configure it first [see below]).

5.6.3 WFS Service Settings

There are two GeoServer WFS service settings that are strongly recommended for interoperable complex feature services. These can be enabled through the *Services* → *WFS* page on the GeoServer web interface or by manually editing the `wfs.xml` file in the data directory,

Canonical schema location

The default GeoServer behaviour is to encode WFS responses that include a `schemaLocation` for the WFS schema that is located on the GeoServer instance. A client will not know without retrieving the schema whether it is identical to the official schema hosted at `schemas.opengis.net`. The solution is to encode the `schemaLocation` for the WFS schema as the canonical location at `schemas.opengis.net`.

To enable this option, choose *one* of these:

1. Either: On the *Service* → *WFS* page under *Conformance* check *Encode canonical WFS schema location*.
2. Or: Insert the following line before the closing tag in `wfs.xml`:

```
<canonicalSchemaLocation>true</canonicalSchemaLocation>
```

Encode using featureMember

By default GeoServer will encode WFS 1.1 responses with multiple features in a single `gml:featureMembers` element. This will cause invalid output if a response includes a feature at the top level that has already been encoded as a nested property of an earlier feature, because there is no single element that can be used to encode this feature by reference. The solution is to encode responses using `gml:featureMember`.

To enable this option, choose *one* of these:

1. Either: On the *Service* → *WFS* page under *Encode response with* select *Multiple "featureMember" elements*.
2. Or: Insert the following line before the closing tag in `wfs.xml`:

```
<encodeFeatureMember>true</encodeFeatureMember>
```

5.6.4 Configuration

Configuration of an app-schema complex feature type requires manual construction of a GeoServer data directory that contains an XML mapping file and a `datastore.xml` that points at this mapping file. The data directory also requires all the other ancillary configuration files used by GeoServer for simple features. GeoServer can serve simple and complex features at the same time.

Workspace layout

The GeoServer data directory contains a folder called `workspaces` with the following structure:

```
workspaces
    - gsml
        - SomeDataStore
            - SomeFeatureType
                - featuretype.xml
            - datastore.xml
            - SomeFeatureType-mapping-file.xml
```

Note: The folder inside `workspaces` must have a name (the workspace name) that is the same as the namespace prefix (`gsml` in this example).

Datastore

Each data store folder contains a file `datastore.xml` that contains the configuration parameters of the data store. To create an app-schema feature type, the data store must be configured to load the app-schema service module and process the mapping file. These options are contained in the `connectionParameters`:

- `namespace` defines the XML namespace of the complex feature type.
- `url` is a `file:` URL that gives the location of the app-schema mapping file relative to the root of the GeoServer data directory.
- `dbtype` must be `app-schema` to trigger the creation of an app-schema feature type.

5.6.5 Mapping File

An app-schema feature type is configured using a mapping file that defines the data source for the feature and the mappings from the source data to XPaths in the output XML.

Outline

Here is an outline of a mapping file:

```
<?xml version="1.0" encoding="UTF-8"?>
<as:AppSchemaDataAccess xmlns:as="http://www.geotools.org/app-schema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.geotools.org/app-schema AppSchemaDataAccess.xsd">
    <namespaces>...</namespaces>
    <includedTypes>...</includedTypes>
    <sourceDataStores>...</sourceDataStores>
    <catalog>...</catalog>
    <targetTypes...</targetTypes>
    <typeMappings>...</typeMappings>
</as:AppSchemaDataAccess>
```

- `namespaces` defines all the namespace prefixes used in the mapping file.
- `includedTypes` (optional) defines all the included non-feature type mapping file locations that are referred in the mapping file.
- `sourceDataStores` provides the configuration information for the source data stores.
- `catalog` is the location of the OASIS Catalog used to resolve XML Schema locations.
- `targetTypes` is the location of the XML Schema that defines the feature type.
- `typeMappings` give the relationships between the fields of the source data store and the elements of the output complex feature.

Mapping file schema

- `AppSchemaDataAccess.xsd` is optional because it is not used by GeoServer. The presence of `AppSchemaDataAccess.xsd` in the same folder as the mapping file enables XML editors to observe its grammar and provide contextual help.

Settings

namespaces

The `namespaces` section defines all the XML namespaces used in the mapping file:

```
<Namespace>
    <prefix>gsml</prefix>
    <uri>urn:cgi:xmlns:CGI:GeoSciML:2.0</uri>
</Namespace>
<Namespace>
    <prefix>gml</prefix>
    <uri>http://www.opengis.net/gml</uri>
</Namespace>
```

```
<Namespace>
    <prefix>xlink</prefix>
    <uri>http://www.w3.org/1999/xlink</uri>
</Namespace>
```

includedTypes (optional)

Non-feature types (eg. gsml:CompositionPart is a data type that is nested in gsml:GeologicUnit) may be mapped separately for its reusability, but we don't want to configure it as a feature type as we don't want to individually access it. Related feature types don't need to be explicitly included here as it would have its own workspace configuration for GeoServer to find it. The location path in `Include` tag is relative to the mapping file. For an example, if gsml:CompositionPart configuration file is located in the same directory as the gsml:GeologicUnit configuration:

```
<includedTypes>
    <Include>gsml_CompositionPart.xml</Include>
</includedTypes>
```

sourceDataStores

Every mapping file requires at least one data store to provide data for features. app-schema reuses GeoServer data stores, so there are many available types. See *Data Stores* for details of data store configuration. For example:

```
<sourceDataStores>
    <DataStore>
        <id>datastore</id>
        <parameters>
            ...
        </parameters>
    </DataStore>
    ...
</sourceDataStores>
```

If you have more than one `DataStore` in a mapping file, be sure to give them each a distinct `id`.

catalog (optional)

The location of an OASIS XML Catalog configuration file, given as a path relative to the mapping file. See *Application Schema Resolution* for more information. For example:

```
<catalog>../../../schemas/catalog.xml</catalog>
```

targetTypes

The `targetTypes` section lists all the application schemas required to define the mapping. Typically only one is required. For example:

```
<targetTypes>
    <FeatureType>
        <schemaUri>http://www.geosciml.org/geosciml/2.0/xsd/geosciml.xsd</schemaUri>
    </FeatureType>
</targetTypes>
```

Mappings

typeMappings and FeatureTypeMapping

The `typeMappings` section is the heart of the app-schema module. It defines the mapping from simple features to the the nested structure of one or more simple features. It consists of a list of `FeatureTypeMapping` elements, which each define one output feature type. For example:

```
<typeMappings>
    <FeatureTypeMapping>
        <mappingName>mappedfeature1</mappingName>
        <sourceDataStore>datastore</sourceDataStore>
        <sourceType>mappedfeature</sourceType>
        <targetElement>gsml:MappedFeature</targetElement>
        <isDenormalised>true</isDenormalised>
        <defaultGeometry>gsml:MappedFeature/gsml:shape/gml:Polygon</defaultGeometry>
        <attributeMappings>
            <AttributeMapping>
                ...
```

- `mappingName` is an optional tag, to identify the mapping in *Feature Chaining* when there are multiple FeatureTypeMapping instances for the same type. This is solely for feature chaining purposes, and would not work for identifying top level features.
- `sourceDataStore` must be an identifier you provided when you defined a source data store the `sourceDataStores` section.
- `sourceType` is the simple feature type name. For example:
  - a table or view name, lowercase for PostGIS, uppercase for Oracle.
  - a property file name (without the .properties suffix)
- `targetElement` is the the element name in the target application schema. This is the same as the WFS feature type name.
- `isDenormalised` is an optional tag (default true) to indicate whether this type contains denormalised data or not. If data is not denormalised, then app-schema will build a more efficient query to apply the global feature limit. When combined with a low global feature limit (via *Services –> WFS*), setting this option to false can prevent unnecessary processing and database lookups from taking place.
- `defaultGeometry` can be used to explicitly define the attribute of the feature type that should be used as the default geometry, this is more relevant in WMS than WFS. The default geometry XML path can reference any attribute of the feature type, exactly the same path that would be used to reference the desired property in a OGC filter. The path can reference a nested attribute belonging to a chained feature having a zero or one relationship with the root feature type.

attributeMappings and AttributeMapping

`attributeMappings` comprises a list of `AttributeMapping` elements:

```
<AttributeMapping>
    <targetAttribute>...</targetAttribute>
    <idExpression>...</idExpression>
    <sourceExpression>...</sourceExpression>
    <targetAttributeNode>...</targetAttributeNode>
    <isMultiple>...</isMultiple>
    <ClientProperty>...</ClientProperty>
</AttributeMapping>
```

targetAttribute

`targetAttribute` is the XPath to the output element, in the context of the target element. For example, if the containing mapping is for a feature, you should be able to map a `gml:name` property by setting the target attribute:

```
<targetAttribute>gml:name</targetAttribute>
```

Multivalued attributes resulting from *Denormalised sources* are automatically encoded. If you wish to encode multivalued attributes from different input columns as a specific instance of an attribute, you can use a (one-based) index. For example, you can set the third `gml:name` with:

```
<targetAttribute>gml:name[3]</targetAttribute>
```

The reserved name `FEATURE_LINK` is used to map data that is not encoded in XML but is required for use in *Feature Chaining*.

idExpression (optional)

A CQL expression that is used to set the custom `gml:id` of the output feature type. This should be the name of a database column on its own. Using functions would cause an exception because it is not supported with the default joining implementation.

Note: Every feature must have a `gml:id`. This requirement is an implementation limitation (strictly, `gml:id` is optional in GML).

- If idExpression is unspecified, `gml:id` will be `<the table name>.<primary key>`, e.g. `MAPPEDFEATURE.1`.
- In the absence of primary keys, this will be `<the table name>.<generated gml id>`, e.g. `MAPPEDFEATURE.fid--46fd41b8_1407138b56f_-7fe0`.
- If using property files instead of database tables, the default `gml:id` will be the row key found before the equals ("=") in the property file, e.g. the feature with row "mf1=Mudstone|POINT(1 2)|..." will have gml:id `mf1`.

Note: `gml:id` must be an NCName.

---

sourceExpression (optional)

Use a `sourceExpression` tag to set the element content from source data. For example, to set the element content from a column called `DESCRIPTION`:

```
<sourceExpression><OCQL>DESCRIPTION</OCQL></sourceExpression>
```

If `sourceExpression` is not present, the generated element is empty (unless set by another mapping).

You can use CQL expressions to calculate the content of the element. This example concatenated strings from two columns and a literal:

```
<sourceExpression>
    <OCQL>strConCat(FIRST , strConCat(' followed by ', SECOND))</OCQL>
</sourceExpression>
```

You can also use *CQL functions* for vocabulary translations.

Warning: Avoid use of CQL expressions for properties that users will want to query, because the current implementation cannot reverse these expressions to generate efficient SQL, and will instead read all features to calculate the property to find the features that match the filter query. Falling back to brute force search makes queries on CQL-calculated expressions very slow. If you must concatenate strings to generate content, you may find that doing this in your database is much faster.

linkElement and linkField (optional)

The presence of `linkElement` and `linkField` change the meaning of `sourceExpression` to a *Feature Chaining* mapping, in which the source of the mapping is the feature of type `linkElement` with property `linkField` matching the expression. For example, the following `sourceExpression` uses as the result of the mapping the (possibly multivalued) `gsml:MappedFeature` for which `gml:name[2]` is equal to the value of `URN` for the source feature. This is in effect a foreign key relation:

```
<sourceExpression>
    <OCQL>URN</OCQL>
    <linkElement>gsml:MappedFeature</linkElement>
    <linkField>gml:name[2]</linkField>
</sourceExpression>
```

The feature type `gsml:MappedFeature` might be defined in another mapping file. The `linkField` can be `FEATURE_LINK` if you wish to relate the features by a property not exposed in XML. See *Feature Chaining* for a comprehensive discussion.

For special cases, `linkElement` could be an OCQL function, and `linkField` could be omitted. See *Polymorphism* for further information.

targetAttributeNode (optional)

`targetAttributeNode` is required wherever a property type contains an abstract element and app-schema cannot determine the type of the enclosed attribute.

In this example, `om:result` is of `xs:anyType`, which is abstract. We can use `targetAttributeNode` to set the type of the property type to a type that encloses a non-abstract element:

```
<AttributeMapping>
      <targetAttribute>om:result</targetAttribute>
      <targetAttributeNode>gml:MeasureType<targetAttributeNode>
      <sourceExpression>
          <OCQL>TOPAGE</OCQL>
      </sourceExpression>
      <ClientProperty>
          <name>xsi:type</name>
          <value>'gml:MeasureType'</value>
      </ClientProperty>
      <ClientProperty>
          <name>uom</name>
          <value>'http://www.opengis.net/def/uom/UCUM/0/Ma'</value>
      </ClientProperty>
</AttributeMapping>
```

If the casting type is complex, the specific type is implicitly determined by the XPath in targetAttribute and targetAttributeNode is not required. E.g., in this example `om:result` is automatically specialised as a MappedFeatureType:

```
<AttributeMapping>
      <targetAttribute>om:result/gsml:MappedFeature/gml:name</targetAttribute>
      <sourceExpression>
          <OCQL>NAME</OCQL>
      </sourceExpression>
</AttributeMapping>
```

Although it is not required, we may still specify targetAttributeNode for the root node, and map the children attributes as per normal. This mapping must come before the mapping for the enclosed elements. By doing this, app-schema will report an exception if a mapping is specified for any of the children attributes that violates the type in targetAttributeNode. E.g.:

```
<AttributeMapping>
      <targetAttribute>om:result</targetAttribute>
      <targetAttributeNode>gsml:MappedFeatureType<targetAttributeNode>
</AttributeMapping>
<AttributeMapping>
      <targetAttribute>om:result/gsml:MappedFeature/gml:name</targetAttribute>
      <sourceExpression>
          <OCQL>NAME</OCQL>
      </sourceExpression>
</AttributeMapping>
```

Note that the GML encoding rules require that complex types are never the direct property of another complex type; they are always contained in a property type to ensure that their type is encoded in a surrounding element. Encoded GML is always type/property/type/property. This is also known as the GML "striping" rule. The consequence of this for app-schema mapping files is that `targetAttributeNode` must be applied to the property and the type must be set to the XSD property type, not to the type of the contained attribute (`gsml:CGI_TermValuePropertyType` not `gsml:CGI_TermValueType`). Because the XPath refers to a property type not the encoded content, `targetAttributeNode` appears in a mapping with `targetAttribute` and no other elements when using with complex types.

The XML encoder will encode nested complex features that are mapped to a complex type that does not respect the GML striping rule. The Java configuration property `encoder.relaxed` can be set to `false` to disable this behavior.

encodeIfEmpty (optional)

The `encodeIfEmpty` element will determine if an attribute will be encoded if it contains a null or empty value. By default `encodeIfEmpty` is set to false therefore any attribute that does not contain a value will be skipped:

```
<encodeIfEmpty>true</encodeIfEmpty>
```

`encodeIfEmpty` can be used to bring up attributes that only contain client properties such as `xlink:title`.

isMultiple (optional)

The `isMultiple` element states whether there might be multiple values for this attribute, coming from denormalised rows. Because the default value is `false` and it is omitted in this case, it is most usually seen as:

```
<isMultiple>true</isMultiple>
```

For example, the table below is denormalised with `NAME` column having multiple values:

ID	NAME	DESCRIPTION
gu.25678	Yaugher Volcanic Group 1	Olivine basalt, tuff, microgabbro
gu.25678	Yaugher Volcanic Group 2	Olivine basalt, tuff, microgabbro

The configuration file specifies `isMultiple` for `gml:name` attribute that is mapped to the `NAME` column:

```
<AttributeMapping>
    <targetAttribute>gml:name</targetAttribute>
    <sourceExpression>
        <OCQL>NAME</OCQL>
    </sourceExpression>
    <isMultiple>true</isMultiple>
    <ClientProperty>
        <name>codeSpace</name>
        <value>'urn:ietf:rfc:2141'</value>
    </ClientProperty>
</AttributeMapping>
```

The output produces multiple `gml:name` attributes for each feature grouped by the id:

```
<gsml:GeologicUnit gml:id="gu.25678">
    <gml:description>Olivine basalt, tuff, microgabbro</gml:description>
    <gml:name codeSpace="urn:ietf:rfc:2141">Yaugher Volcanic Group 1</gml:name>
    <gml:name codeSpace="urn:ietf:rfc:2141">Yaugher Volcanic Group 2</gml:name>
 ...
```

isList (optional)

The `isList` element states whether there might be multiple values for this attribute, concatenated as a list. The usage is similar with `isMultiple`, except the values appear concatenated inside a single node instead of each value encoded in a separate node. Because the default value is `false` and it is omitted in this case, it is most usually seen as:

```
<isList>true</isList>
```

For example, the table below has multiple `POSITION` for each feature:

ID	POSITION
ID1.2	1948-05
ID1.2	1948-06
ID1.2	1948-07
ID1.2	1948-08
ID1.2	1948-09

The configuration file uses `isList` on `timePositionList` attribute mapped to `POSITION` column:

```
<AttributeMapping>
    <targetAttribute>csml:timePositionList</targetAttribute>
    <sourceExpression>
        <OCQL>POSITION</OCQL>
    </sourceExpression>
    <isList>true</isList>
</AttributeMapping>
```

The output produced:

```
<csml:pointSeriesDomain>
    <csml:TimeSeries gml:id="ID1.2">
        <csml:timePositionList>1949-05 1949-06 1949-07 1949-08 1949-09</
↪csml:timePositionList>
    </csml:TimeSeries>
</csml:pointSeriesDomain>
```

ClientProperty (optional, multivalued)

A mapping can have one or more `ClientProperty` elements which set XML attributes on the mapping target. Each `ClientProperty` has a `name` and a `value` that is an arbitrary CQL expression. No `OCQL` element is used inside `value`.

This example of a `ClientProperty` element sets the `codeSpace` XML attribute to the literal string `urn:ietf:rfc:2141`. Note the use of single quotes around the literal string. This could be applied to any target attribute of GML CodeType:

```
<ClientProperty>
    <name>codeSpace</name>
    <value>'urn:ietf:rfc:2141'</value>
</ClientProperty>
```

When the GML association pattern is used to encode a property by reference, the `xlink:href` attribute is set and the element is empty. This `ClientProperty` element sets the `xlink:href` XML attribute to to the value of the `RELATED_FEATURE_URN` field in the data source (for example, a column in an Oracle database table). This mapping could be applied to any property type, such a `gml:FeaturePropertyType`, or other type modelled on the GML association pattern:

```
<ClientProperty>
    <name>xlink:href</name>
```

```
    <value>RELATED_FEATURE_URN</value>
</ClientProperty>
```

See the discussion in *Feature Chaining* for the special case in which `xlink:href` is created for multivalued properties by reference.

CQL

CQL functions enable data conversion and conditional behaviour to be specified in mapping files.

- See *CQL functions* for information on additional functions provided by the app-schema plugin.
- The uDig manual includes a list of CQL functions:
  - http://udig.refractions.net/confluence/display/EN/Constraint+Query+Language
- CQL string literals are enclosed in single quotes, for example `'urn:ogc:def:nil:OGC:missing'`.

Database identifiers

When referring to database table/view names or column names, use:

- lowercase for PostGIS
- UPPERCASE for Oracle Spatial and ArcSDE

Denormalised sources

Multivalued properties from denormalised sources (the same source feature ID appears more than once) are automatically encoded. For example, a view might have a repeated `id` column with varying `name` so that an arbitrarily large number of `gml:name` properties can be encoded for the output feature.

Warning: Denormalised sources must grouped so that features with duplicate IDs are provided without any intervening features. This can be achieved by ensuring that denormalised source features are sorted by ID. Failure to observe this restriction will result in data corruption. This restriction is however not necessary when using *Joining Support For Performance* because then ordering will happen automatically.

Attributes with cardinality 1..N

Consider the following two tables, the first table contains information related to meteorological stations:

ID	NAME
st.1	Station 1
st.2	Station 2

The second table contains tags that are associated with meteorological stations:

ID	STATION_ID	TAG	CODE
tg.1	st.1	temperature	X1Y
tg.2	st.1	wind	X2Y
tg.2	st.2	pressure	X3Y

A station can have multiple tags, establishing a one to many relationship between stations and tags, the GML representation of the first station should look like this:

```
(...)
<st:Station gml:id="st.1">
  <st:name>Station 1</st:name>
  <st:tag st:code="X1Y">temperature</st:tag>
  <st:tag st:code="X2Y">wind</st:tag>
</st:Station_gml32>
(...)
```

Mappings with a one to many relationship are supported with a custom syntax in JDBC based data stores and Apache Solr data store.

SQL based data stores

When using JDBC based data stores attributes with a 1..N relationship can be mapped like this:

```
(...)
<AttributeMapping>
  <targetAttribute>st:tag</targetAttribute>
  <jdbcMultipleValue>
    <sourceColumn>ID</sourceColumn>
    <targetTable>TAGS</targetTable>
    <targetColumn>STATION_ID</targetColumn>
    <targetValue>TAG</targetValue>
  </jdbcMultipleValue>
  <ClientProperty>
    <name>st:code</name>
    <value>CODE</value>
  </ClientProperty>
</AttributeMapping>
(...)
```

The `targetValue` refers to the value of the `<st:tag>` element, the client property is mapped with the usual syntax. Behind the scenes App-Schema will take care of associating the `st:code` attribute value with the correct tag.

Apache Solr

When using Apache Solr data stores, 1..N relationships can be handled with `multiValued` fields and mapped like this:

```
(...)
<AttributeMapping>
  <targetAttribute>st:tag</targetAttribute>
  <solrMultipleValue>TAG</solrMultipleValue>
  <ClientProperty>
    <name>st:code</name>
    <value>CODE</value>
  </ClientProperty>
</AttributeMapping>
(...)
```

External Apache Solr Index

Is possible to use an external Apache Solr index to speed up text based searches. Consider the following WFS `GetFeatureRequest`:

```
<wfs:GetFeature service="WFS" version="2.0.0" xmlns:fes="http://www.opengis.net/fes/2.
↪0" xmlns:gml="http://www.opengis.net/gml/3.2.1" xmlns:wfs="http://www.opengis.net/
↪wfs/2.0">
  <wfs:Query typeNames="st:Station">
    <fes:Filter>
      <fes:PropertyIsLike escapeChar="!" singleChar="." wildCard="*">
        <fes:ValueReference>st:Station/st:measurement/st:Measurement/st:description</
↪fes:ValueReference>
        <fes:Literal>*high*</fes:Literal>
      </fes:PropertyIsLike>
    </fes:Filter>
  </wfs:Query>
</wfs:GetFeature>
```

This request will return all the stations that have at least one measurement that contains the text `high` in its description. This type of text based queries are (usually) quite expensive to execute in relational data bases.

Apache Solr is a well known open source project that aims to solve those type of performance issues, it allow us to index several fields and efficiently query those fields. Although Apache Solr allow us to index several types of fields, e.g. numbers or even latitudes longitudes, where it really shines is when dealing with text fields and text based queries.

The goal of external indexes is to allow App-Schema to take advantage of Apache Solr text searches performance and at the same time to take advantage of relational databases relational capabilities. In practice, this means that the full data will be stored in the relational database and we will only need to index in Apache Solr the fields we need to improve our text based queries.

Using the stations use case as an example, our data will be stored in a relational database, e.g. PostgreSQL, and we will index the possible descriptions measurements values in an Apache Solr index.

Our mapping file will look like this:

```
<as:AppSchemaDataAccess xmlns:as="http://www.geotools.org/app-schema" xmlns:xsi=
↪"http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.geotools.
↪org/app-schema AppSchemaDataAccess.xsd">
  <namespaces>
    <Namespace>
      <prefix>gml</prefix>
      <uri>http://www.opengis.net/gml/3.2</uri>
    </Namespace>
    <Namespace>
      <prefix>st</prefix>
      <uri>http://www.stations.org/1.0</uri>
    </Namespace>
  </namespaces>
  <sourceDataStores>
    <SolrDataStore>
      <id>stations_solr</id>
      <url>http://localhost:8983/solr/stations_index</url>
      <index name="stations_index"/>
    </SolrDataStore>
    <DataStore>
      <id>stations_db</id>
      <parameters>
```

```
        <Parameter>
          <name>dbtype</name>
          <value>postgisng</value>
        </Parameter>
      </parameters>
    </DataStore>
  </sourceDataStores>
  <targetTypes>
    <FeatureType>
      <schemaUri>./stations.xsd</schemaUri>
    </FeatureType>
  </targetTypes>
  <typeMappings>
    <FeatureTypeMapping>
      <sourceDataStore>stations_db</sourceDataStore>
      <sourceType>stations</sourceType>
      <targetElement>st:Station</targetElement>
      <!-- configure the index data store for this feature type -->
      <indexDataStore>stations_solr</indexDataStore>
      <indexType>stations_index</indexType>
      <attributeMappings>
        <AttributeMapping>
          <targetAttribute>st:Station</targetAttribute>
          <idExpression>
            <OCQL>id</OCQL>
          </idExpression>
          <!-- the Solr index field that matches this feature type table primary key -
↪->
          <indexField>id</indexField>
        </AttributeMapping>
        <AttributeMapping>
          <targetAttribute>st:name</targetAttribute>
          <sourceExpression>
            <OCQL>name</OCQL>
          </sourceExpression>
        </AttributeMapping>
        <AttributeMapping>
          <targetAttribute>st:measurement</targetAttribute>
          <sourceExpression>
            <OCQL>id</OCQL>
            <linkElement>st:Measurement</linkElement>
            <linkField>FEATURE_LINK[1]</linkField>
          </sourceExpression>
          <isMultiple>true</isMultiple>
        </AttributeMapping>
      </attributeMappings>
    </FeatureTypeMapping>
    <FeatureTypeMapping>
      <sourceDataStore>stations_db</sourceDataStore>
      <sourceType>measurements</sourceType>
      <targetElement>st:Measurement</targetElement>
      <attributeMappings>
        <AttributeMapping>
          <targetAttribute>st:Measurement</targetAttribute>
          <idExpression>
            <OCQL>id</OCQL>
          </idExpression>
        </AttributeMapping>
```

```
        <AttributeMapping>
          <targetAttribute>st:description</targetAttribute>
          <sourceExpression>
            <OCQL>description</OCQL>
          </sourceExpression>
          <!-- the Solr index field that indexes the description possible values -->
          <indexField>description_txt</indexField>
        </AttributeMapping>
        <AttributeMapping>
          <targetAttribute>FEATURE_LINK[1]</targetAttribute>
          <sourceExpression>
            <OCQL>station_id</OCQL>
          </sourceExpression>
        </AttributeMapping>
      </attributeMappings>
    </FeatureTypeMapping>
  </typeMappings>
</as:AppSchemaDataAccess>
```

To be able to use an external Apache Solr index, we need at least to:

- **declare the Solr data store and the index**: this is done in the root feature type mapping, e.g. *st:Station*.
- **map the Solr index field that matches the database primary key**: this is done id mapping of the root feature type, e.g."<indexField>id</indexField>".
- **map each attribute that is indexed in Apache Solr**: this is done using the *indexField* element, e.g `<indexField>description_txt</indexField>`.

Is worth mentioning that if an external Solr index was defined, App-Schema will always query the external Solr index first and then query the relational database.

5.6.6 Application Schema Resolution

To be able to encode XML responses conforming to a GML application schema, the app-schema plugin must be able to locate the application schema files (XSDs) that define the schema. This page describes the schema resolution process.

Schema downloading is now automatic for most users

GeoServer will automatically download and cache (see *Cache* below) all the schemas it needs the first time it starts if:

1. All the application schemas you use are accessed via http/https URLs, and
2. Your GeoServer instance is deployed on a network that permits it to download them.

Note: This is the recommended way of using GeoServer app-schema for most users.

If cached downloading is used, no manual handling of schemas will be required. The rest of this page is for those with more complicated arrangements, or who wish to clear the cache.

Resolution order

The order of sources used to resolve application schemas is:

1. *OASIS Catalog*
2. *Classpath*
3. *Cache*

Every attempt to load a schema works down this list, so imports can be resolved from sources other than that used for the originating document. For example, an application schema in the cache that references a schema found in the catalog will use the version in the catalog, rather than caching it. This allows users to supply unpublished or modified schemas sourced from, for example, the catalog, at the cost of interoperability (how do WFS clients get them?).

OASIS Catalog

An OASIS XML Catalog is a standard configuration file format that instructs an XML processing system how to process entity references. The GeoServer app-schema resolver uses catalog URI semantics to locate application schemas, so `uri` or `rewriteURI` entries should be present in your catalog. The optional mapping file `catalog` element provides the location of the OASIS XML Catalog configuration file, given as a path relative to the mapping file, for example:

```
<catalog>../../../schemas/catalog.xml</catalog>
```

Earlier versions of the app-schema plugin required all schemas to be present in the catalog. This is no longer the case. Because the catalog is searched first, existing catalog-based deployments will continue to work as before.

To migrate an existing GeoServer app-schema deployment that uses an OASIS Catalog to instead use cached downloads (see *Cache* below), remove all `catalog` elements from your mapping files and restart GeoServer.

Classpath

Java applications such as GeoServer can load resources from the Java classpath. GeoServer app-schema uses a simple mapping from an http or https URL to a classpath resource location. For example, an application schema published at `http://schemas.example.org/exampleml/exml.xsd` would be found on the classpath if it was stored either:

- at `/org/example/schemas/exampleml/exml.xsd` in a JAR file on the classpath (for example, a JAR file in `WEB-INF/lib`) or,
- on the local filesystem at `WEB-INF/classes/org/example/schemas/exampleml/exml.xsd`.

The ability to load schemas from the classpath is intended to support testing, but may be useful to users whose communities supply JAR files containing their application schemas.

Cache

If an application schema cannot be found in the catalog or on the classpath, it is downloaded from the network and stored in a subdirectory `app-schema-cache` of the GeoServer data directory.

- Once schemas are downloaded into the cache, they persist indefinitely, including over GeoServer restarts.
- No attempt will be made to retrieve new versions of cached schemas.
- To clear the cache, remove the subdirectory `app-schema-cache` of the GeoServer data directory and restart GeoServer.

GeoServer app-schema uses a simple mapping from an http or https URL to local filesystem path. For example, an application schema published at `http://schemas.example.org/exampleml/exml.xsd` would be downloaded and stored as `app-schema-cache/org/example/schemas/exampleml/exml.xsd`. Note that:

- Only `http` and `https` URLs are supported.
- Port numbers, queries, and fragments are ignored.

If your GeoServer instance is deployed on a network whose firewall rules prevent outgoing TCP connections on port 80 (http) or 443 (https), schema downloading will not work. (For security reasons, some service networks ["demilitarised zones"] prohibit such outgoing connections.) If schema downloading is not permitted on your network, there are three solutions:

1. Either: Install and configure GeoServer on another network that can make outgoing TCP connections, start GeoServer to trigger schema download, and then manually copy the `app-schema-cache` directory to the production server. This is the easiest option because GeoServer automatically downloads all the schemas it needs, including dependencies.
2. Or: Deploy JAR files containing all required schema files on the classpath (see *Classpath* above).
3. Or: Use a catalog (see *OASIS Catalog* above).

Warning: System property "schema.cache.dir" with a cache directory location is required for using a mapping file from a remote URL with 'http://' or 'https://' protocol.

5.6.7 Supported GML Versions

GML 3.1.1

- GML 3.1.1 application schemas are supported for WFS 1.1.0.
- Clients must specify WFS 1.1.0 in requests because the GeoServer default is WFS 2.0.0.
- GET URLs must contain `version=1.1.0` to set the WFS version to 1.1.0.

GML 3.2.1

- GML 3.2.1 application schemas are supported for WFS 1.1.0 and (incomplete) WFS 2.0.0.
- Some WFS 2.0.0 features not in WFS 1.1.0 such as GetFeatureById are not yet supported.
- Clients using WFS 1.1.0 must specify WFS 1.1.0 in requests and select the `gml32` output format for GML 3.2.1.
- To use WFS 1.1.0 for GML 3.2.1, GET URLs must contain `version=1.1.0` to set the WFS version to 1.1.0 and `outputFormat=gml32` to set the output format to GML 3.2.1.
- The default WFS version is 2.0.0, for which the default output format is GML 3.2.1.
- All GML 3.2.1 responses are contained in a WFS 2.0.0 `FeatureCollection` element, even for WFS 1.1.0 requests, because a WFS 1.1.0 `FeatureCollection` cannot contain GML 3.2.1 features.

Secondary namespace for GML 3.2.1 required

GML 3.2.1 WFS responses are delivered in a WFS 2.0.0 `FeatureCollection`. Unlike WFS 1.1.0, WFS 2.0.0 does not depend explicitly on any GML version. As a consequence, the GML namespace is secondary and must be defined explicitly as a secondary namespace. See *Secondary Namespaces* for details.

For example, to use the prefix `gml` for GML 3.2, create `workspaces/gml/namespace.xml` containing:

```
<namespace>
    <id>gml_namespace</id>
    <prefix>gml</prefix>
    <uri>http://www.opengis.net/gml/3.2</uri>
</namespace>
```

and `workspaces/gml/workspace.xml` containing:

```
<workspace>
    <id>gml_workspace</id>
    <name>gml</name>
</workspace>
```

Failure to define the `gml` namespace prefix with a secondary namespace will result in errors like:

```
java.io.IOException: The prefix "null" for element "null:name" is not bound.
```

while encoding a response (in this case one containing `gml:name`), even if the namespace prefix is defined in the mapping file.

GML 3.2.1 geometries require gml:id

GML 3.2.1 requires that all geometries have a `gml:id`. While GeoServer will happily encode WFS responses without `gml:id` on geometries, these will be schema-invalid. Encoding a `gml:id` on a geometry can be achieved by setting an `idExpression` in the mapping for the geometry property. For example, `gsml:shape` is a geometry property and its `gml:id` might be generated with:

```
<AttributeMapping>
    <targetAttribute>gsml:shape</targetAttribute>
    <idExpression>
        <OCQL>strConcat('shape.', getId())</OCQL>
    </idExpression>
    <sourceExpression>
        <OCQL>SHAPE</OCQL>
    </sourceExpression>
</AttributeMapping>
```

In this example, `getId()` returns the `gml:id` of the containing feature, so each geometry will have a unique `gml:id` formed by appending the `gml:id` of the containing feature to the string `"shape."`.

If a multigeometry (such as a `MultiPoint` or `MultiSurface`) is assigned a `gml:id` of (for example) `parentid`, to permit GML 3.2.1 schema-validity, each geometry that the multigeometry contains will be automatically assigned a `gml:id` of the form `parentid.1`, `parentid.2`, ... in order.

GML 3.3

The proposed GML 3.3 is itself a GML 3.2.1 application schema; preliminary testing with drafts of GML 3.3 indicates that it works with app-schema as expected.

5.6.8 Secondary Namespaces

What is a secondary namespace?

A secondary namespace is one that is referenced indirectly by the main schema, that is, one schema imports another one as shown below:

```
a.xsd imports b.xsd
b.xsd imports c.xsd
```

(using a, b and c as the respective namespace prefixes for a.xsd, b.xsd and c.xsd):

```
a.xsd declares b:prefix
b.xsd declares c:prefix
```

The GeoTools encoder does not honour these namespaces and writes out:

```
"a:" , "b:" but NOT "c:"
```

The result is c's element being encoded as:

```
<null:cElement/>
```

When to configure for secondary namespaces

If your application spans several namespaces which may be very common in application schemas.

A sure sign that calls for secondary namespace configuration is when prefixes for namespaces are printed out as the literal string "null" or error messages like:

```
java.io.IOException: The prefix "null" for element "null:something" is not bound.
```

Note: When using secondary namespaces, requests involving complex featuretypes must be made to the **global OWS service** only, not to *Virtual Services*. This is because virtual services are restricted to a single namespace, and thus are not able to access secondary namespaces.

In order to allow GeoServer App-Schema to support secondary namespaces, please follow the steps outlined below:

Using the sampling namespace as an example.

Step 1:Create the Secondary Namespace folder

Create a folder to represent the secondary namespace in the data/workspaces directory, in our example that will be the "sa" folder.

Step 2:Create files

Create two files below in the "sa" folder:

1. namespace.xml
2. workspace.xml

Step 3:Edit content of files

Contents of these files are as follows:

namespace.xml(uri is a valid uri for the secondary namespace, in this case the sampling namespace uri):

```
<namespace>
        <id>sa_workspace</id>
        <prefix>sa</prefix>
        <uri>http://www.opengis.net/sampling/1.0</uri>
</namespace>
```

workspace.xml:

```
<workspace>
        <id>sa_workspace</id>
        <name>sa</name>
</workspace>
```

That's it.

Your workspace is now configured to use a Secondary Namespace.

5.6.9 CQL functions

CQL functions enable data conversion and conditional behaviour to be specified in mapping files. Some of these functions are provided by the app-schema plugin specifically for this purpose.

- The uDig manual includes a list of CQL functions:
  - http://udig.refractions.net/confluence/display/EN/Constraint%20Query%20Language.html
- CQL string literals are enclosed in single quotes, for example `'urn:ogc:def:nil:OGC:missing'`.
- Single quotes are represented in CQL string literals as two single quotes, just as in SQL. For example, `'yyyy-MM-dd''T''HH:mm:ss''Z'''` for the string `yyyy-MM-dd'T'HH:mm:ss'Z'`.

Vocabulary translation

This section describes how to serve vocabulary translations using some function expressions in application schema mapping file. If you're not familiar with application schema mapping file, read *Mapping File*.

Recode

This is similar to *if_then_else* function, except that there is no default clause. You have to specify a translation value for every vocabulary key.

Syntax:

```
Recode(COLUMN_NAME, key1, value1, key2, value2,...)
```

- **COLUMN_NAME**: column name to get values from

Example:

```
<AttributeMapping>
  <targetAttribute>gml:name</targetAttribute>
  <sourceExpression>
      <OCQL>Recode(ABBREVIATION, '1GRAV',
↪'urn:cgi:classifier:CGI:SimpleLithology:2008:gravel',
                                '1TILL',
↪'urn:cgi:classifier:CGI:SimpleLithology:2008:diamictite',
                                '6ALLU',
↪'urn:cgi:classifier:CGI:SimpleLithology:2008:sediment')
      </OCQL>
  </sourceExpression>
</AttributeMapping>
```

The above example will map **gml:name** value to *urn:cgi:classifier:CGI:SimpleLithology:2008:gravel* if the ABBREVIATION column value is *1GRAV*.

Categorize

This is more suitable for numeric keys, where the translation value is determined by the key's position within the thresholds.

Syntax:

```
Categorize(COLUMN_NAME, default_value, threshold 1, value 1, threshold 2, value 2, ...
↪, [preceding/succeeding])
```

- **COLUMN_NAME**: data source column name
- **default_value**: default value to be mapped if COLUMN_NAME value is not within the threshold
- **threshold(n)**: threshold value
- **value(n)**: value to be mapped if the threshold is met
- **preceding/succeeding:**
  - optional, succeeding is used by default if not specified.
  - not case sensitive.
  - preceding: value is within threshold if COLUMN_NAME value > threshold
  - succeeding: value is within threshold if COLUMN_NAME value >= threshold

Example:

```
<AttributeMapping>
  <targetAttribute>gml:description</targetAttribute>
  <sourceExpression>
      <OCQL>Categorize(CGI_LOWER_RANGE, 'missing_value', 1000, 'minor', 5000,
↪'significant')</OCQL>
  </sourceExpression>
</AttributeMapping>
```

The above example means **gml:description** value would be *significant* if CGI_LOWER_RANGE column value is >= *5000*.

Vocab

This function is more useful for bigger vocabulary pairs. Instead of writing a long key-to-value pairs in the function, you can keep them in a separate properties file. The properties file serves as a lookup table to the function. It has no header, and only contains the pairs in "<key>=<value>" format.

Syntax:

```
Vocab(COLUMN_NAME, properties file)
```

- **COLUMN_NAME**: column name to get values from
- **properties file**: absolute path of the properties file

Example:

Properties file:

```
1GRAV=urn:cgi:classifier:CGI:SimpleLithology:2008:gravel
1TILL=urn:cgi:classifier:CGI:SimpleLithology:2008:diamictite
6ALLU=urn:cgi:classifier:CGI:SimpleLithology:2008:sediment
```

Mapping file:

```
<AttributeMapping>
  <targetAttribute>gml:name</targetAttribute>
  <sourceExpression>
      <OCQL>Vocab(ABBREVIATION, strconcat('${config.parent}', '/mapping.properties'))
↪</OCQL>
  </sourceExpression>
</AttributeMapping>
```

The above example will map **gml:name** to *urn:cgi:classifier:CGI:SimpleLithology:2008:gravel* if ABBREVIATION value is *1GRAV*.

This example uses the `config.parent` predefined interpolation property to specify a vocabulary properties file in the same directory as the mapping file. See *Property Interpolation* for details.

Geometry creation

toDirectPosition

This function converts double values to `DirectPosition` geometry type. This is needed when the data store doesn't have geometry type columns. This function expects:

Literal `'SRS_NAME'` (optional)

Expression expression of SRS name if `'SRS_NAME'` is present as the first argument

Expression name of column pointing to first double value

Expression name of column pointing to second double value (optional, only for 2D)

ToEnvelope

`ToEnvelope` function can take in the following set of parameters and return as either `Envelope` or `ReferencedEnvelope` type:

Option 1 (1D Envelope):

```
ToEnvelope(minx,maxx)
```

Option 2 (1D Envelope with crsname):

```
ToEnvelope(minx,maxx,crsname)
```

Option 3 (2D Envelope):

```
ToEnvelope(minx,maxx,miny,maxy)
```

Option 4 (2D Envelope with crsname):

```
ToEnvelope(minx,maxx,miny,maxy,crsname)
```

toPoint

This function converts double values to a 2D Point geometry type. This is needed when the data store doesn't have geometry type columns. This function expects:

Literal `'SRS_NAME'` (optional)

Expression expression of SRS name if `'SRS_NAME'` is present as the first argument

Expression name of column pointing to first double value

Expression name of column pointing to second double value

Expression expression of gml:id (optional)

toLineString

This function converts double values to 1D LineString geometry type. This is needed to express 1D borehole intervals with custom (non EPSG) CRS.

Literal `'SRS_NAME'` (EPSG code or custom SRS)

Expression name of column pointing to first double value

Expression name of column pointing to second double value

Reference

toXlinkHref

This function redirects an attribute to be encoded as xlink:href, instead of being encoded as a full attribute. This is useful in polymorphism, where static client property cannot be used when the encoding is conditional. This function expects:

Expression REFERENCE_VALUE (could be another function or literal)

Date/time formatting

FormatDateTimezone

A function to format a date/time using a SimpleDateFormat pattern in a time zone supported by Java. This function improves on `dateFormat`, which formats date/time in the server time zone and can produce unintended results. Note that the term "date" is derived from a Java class name; this class represents a date/time, not just a single day.

Syntax:

```
FormatDateTimezone(pattern, date, timezone)
```

pattern formatting pattern supported by SimpleDateFormat, for example `'yyyy-MM-dd'`. Use two single quotes to include a literal single quote in a CQL string literal, for example `'yyyy-MM-dd''T''HH:mm:ss''Z'''`.

date the date/time to be formatted or its string representation, for example `'1948-01-01T00:00:00Z'`. An exception will be returned if the date is malformed (and not null). Database types with time zone information are recommended.

timezone the name of a time zone supported by Java, for example `'UTC'` or `'Canada/Mountain'`. Note that unrecognised timezones will silently be converted to UTC.

This function returns null if any parameter is null.

This example formats date/times from a column `POSITION` in UTC for inclusion in a `csml:TimeSeries`:

```
<AttributeMapping>
    <targetAttribute>csml:timePositionList</targetAttribute>
    <sourceExpression>
        <OCQL>FormatDateTimezone('yyyy-MM-dd''T''HH:mm:ss''Z''', POSITION, 'UTC')</
↪OCQL>
    </sourceExpression>
    <isList>true</isList>
</AttributeMapping>
```

5.6.10 Property Interpolation

Interpolation in this context means the substitution of variables into strings. GeoServer app-schema supports the interpolation of properties (the Java equivalent of environment variables) into app-schema mapping files. This can be used, for example, to simplify the management of database connection parameters that would otherwise be hardcoded in a particular mapping file. This enables data directories to be given to third parties without inapplicable authentication or system configuration information. Externalising these parameters make management easier.

Defining properties

- If the system property `app-schema.properties` is not set, properties are loaded from `WEB-INF/classes/app-schema.properties` (or another resource `/app-schema.properties` on the classpath).
- If the system property `app-schema.properties` is set, properties are loaded from the file named as the value of the property. This is principally intended for debugging, and is designed to be used in an Eclipse launch configuration.

  - For example, if the JVM is started with `-Dapp-schema.properties=/path/to/some/local.properties`, properties are loaded from `/path/to/some/local.properties`.

- System properties override properties defined in a configuration file, so if you define `-Dsome.property` at the java command line, it will override a value specified in the `app-schema.properties` file. This is intended for debugging, so you can set a property file in an Eclipse launch configuration, but override some of the properties contained in the file by setting them explicitly as system properties.
- All system properties are available for interpolation in mapping files.

Predefined properties

If not set elsewhere, the following properties are set for each mapping file:

- `config.file` is set to the name of the mapping file
- `config.parent` is set to the name of the directory containing the mapping file

Using properties

- Using `${some.property}` anywhere in the mapping file will cause it to be replaced by the value of the property `some.property`.
- It is an error for a property that has not been set to be used for interpolation.
- Interpolation is performed repeatedly, so values can contain new interpolations. Use this behaviour with caution because it may cause an infinite loop.
- Interpolation is performed before XML parsing, so can be used to include arbitrary chunks of XML.

Example of property interpolation

This example defines an Oracle data store, where the connection parameter are interpolated from properties:

```
<sourceDataStores>
    <DataStore>
        <id>datastore</id>
        <parameters>
            <Parameter>
                <name>dbtype</name>
                <value>Oracle</value>
            </Parameter>
            <Parameter>
                <name>host</name>
                <value>${example.host}</value>
            </Parameter>
            <Parameter>
                <name>port</name>
                <value>1521</value>
            </Parameter>
            <Parameter>
                <name>database</name>
                <value>${example.database}</value>
            </Parameter>
            <Parameter>
```

```
                <name>user</name>
                <value>${example.user}</value>
            </Parameter>
            <Parameter>
                <name>passwd</name>
                <value>${example.passwd}</value>
            </Parameter>
        </parameters>
    </DataStore>
</sourceDataStores>
```

Example property file

This sample property file gives the property values that are interpolated into the mapping file fragment above. These properties can be installed in `WEB-INF/classes/app-schema.properties` in your GeoServer installation:

```
example.host = database.example.com
example.database = example
example.user = dbuser
example.passwd = s3cr3t
```

5.6.11 Data Stores

The app-schema *Mapping File* requires you to specify your data sources in the `sourceDataStores` section. For GeoServer simple features, these are configured using the web interface, but because app-schema lacks a web configuration interface, data stores must be configured by editing the mapping file.

Many configuration options may be externalised through the use of *Property Interpolation*.

The DataStore element

A `DataStore` configuration consists of

- an `id`, which is an opaque identifier used to refer to the data store elsewhere in a mapping file, and
- one or more `Parameter` elements, which each contain the `name` and `value` of one parameter, and are used to configure the data store.

An outline of the `DataStore` element:

```
<DataStore>
     <id>datastore</id>
     <parameters>
         <Parameter>
             <name>...</name>
             <value>...</value>
         </Parameter>
         ...
     </parameters>
</DataStore>
```

Parameter order is not significant.

Database options

Databases such as PostGIS, Oracle, and ArcSDE share some common or similar configuration options.

`name`	Meaning	`value` examples
`dbtype`	Database type	`postgisng`, `Oracle`, `arcsde`
`host`	Host name or IP address of database server	`database.example.org`, `192.168.3.12`
`port`	TCP port on database server	Default if omitted: `1521` (Oracle), `5432` (PostGIS), `5151` (ArcSDE)
`database`	PostGIS/Oracle database	
`instance`	ArcSDE instance	
`schema`	The database schema	
`user`	The user name used to login to the database server	
`passwd`	The password used to login to the database server	
`Expose primary keys`	Columns with primary keys available for mapping	Default is `false`, set to `true` to use primary key columns in mapping

PostGIS

Set the parameter `dbtype` to `postgisng` to use the PostGIS NG (New Generation) driver bundled with GeoServer 2.0 and later.

Example:

```
<DataStore>
    <id>datastore</id>
    <parameters>
        <Parameter>
            <name>dbtype</name>
            <value>postgisng</value>
        </Parameter>
        <Parameter>
            <name>host</name>
            <value>postgresql.example.org</value>
        </Parameter>
        <Parameter>
            <name>port</name>
            <value>5432</value>
        </Parameter>
        <Parameter>
            <name>database</name>
            <value>test</value>
        </Parameter>
        <Parameter>
            <name>user</name>
            <value>test</value>
        </Parameter>
        <Parameter>
            <name>passwd</name>
            <value>test</value>
        </Parameter>
```

```
    </parameters>
</DataStore>
```

Note: PostGIS support is included in the main GeoServer bundle, so a separate plugin is not required.

Oracle

Set the parameter `dbtype` to `Oracle` to use the Oracle Spatial NG (New Generation) driver compatible with GeoServer 2.0 and later.

Example:

```
<DataStore>
    <id>datastore</id>
    <parameters>
        <Parameter>
            <name>dbtype</name>
            <value>Oracle</value>
        </Parameter>
        <Parameter>
            <name>host</name>
            <value>oracle.example.org</value>
        </Parameter>
        <Parameter>
            <name>port</name>
            <value>1521</value>
        </Parameter>
        <Parameter>
            <name>database</name>
            <value>demodb</value>
        </Parameter>
        <Parameter>
            <name>user</name>
            <value>orauser</value>
        </Parameter>
        <Parameter>
            <name>passwd</name>
            <value>s3cr3t</value>
        </Parameter>
    </parameters>
</DataStore>
```

Note: You must install the Oracle plugin to connect to Oracle Spatial databases.

ArcSDE

This example connects to an ArcSDE database:

```
<DataStore>
    <id>datastore</id>
    <parameters>
        <Parameter>
```

```
            <name>dbtype</name>
            <value>arcsde</value>
        </Parameter>
        <Parameter>
            <name>server</name>
            <value>arcsde.example.org</value>
        </Parameter>
        <Parameter>
            <name>port</name>
            <value>5151</value>
        </Parameter>
        <Parameter>
            <name>instance</name>
            <value>sde</value>
        </Parameter>
        <Parameter>
            <name>user</name>
            <value>demo</value>
        </Parameter>
        <Parameter>
            <name>password</name>
            <value>s3cr3t</value>
        </Parameter>
        <Parameter>
            <name>datastore.allowNonSpatialTables</name>
            <value>true</value>
        </Parameter>
    </parameters>
</DataStore>
```

The use of non-spatial tables aids delivery of application schemas that use non-spatial properties.

Note: You must install the ArcSDE plugin to connect to ArcSDE databases.

Shapefile

Shapefile data sources are identified by the presence of a parameter `url`, whose value should be the file URL for the .shp file.

In this example, only the `url` parameter is required. The others are optional:

```
<DataStore>
    <id>shapefile</id>
    <parameters>
        <Parameter>
            <name>url</name>
            <value>file:/D:/Workspace/shapefiles/VerdeRiverBuffer.shp</value>
        </Parameter>
        <Parameter>
            <name>memory mapped buffer</name>
            <value>false</value>
        </Parameter>
        <Parameter>
            <name>create spatial index</name>
            <value>true</value>
```

```
        </Parameter>
        <Parameter>
            <name>charset</name>
            <value>ISO-8859-1</value>
        </Parameter>
    </parameters>
</DataStore>
```

Note: The `url` in this case is an example of a Windows filesystem path translated to URL notation.

Note: Shapefile support is included in the main GeoServer bundle, so a separate plugin is not required.

Property file

Property files are configured by specifying a `directory` that is a `file:` URI.

- If the directory starts with `file:./` it is relative to the mapping file directory. (This is an invalid URI, but it works.)

For example, the following data store is used to access property files in the same directory as the mapping file:

```
<DataStore>
    <id>propertyfile</id>
    <parameters>
        <Parameter>
            <name>directory</name>
            <value>file:./</value>
        </Parameter>
    </parameters>
</DataStore>
```

A property file data store contains *all* the feature types stored in .properties files in the directory. For example, if the directory contained River.properties and station.properties, the data store would be able to serve them as the feature types `River` and `station`. Other file extensions are ignored.

Note: Property file support is included in the main GeoServer bundle, so a separate plugin is not required.

JNDI

Defining a JDBC data store with a `jndiReferenceName` allows you to use a connection pool provided by your servlet container. This allows detailed configuration of connection pool parameters and sharing of connections between data sources, and even between servlets.

To use a JNDI connection provider:

1. Specify a `dbtype` parameter to to indicate the database type. These values are the same as for the non-JNDI examples above.
2. Give the `jndiReferenceName` you set in your servlet container. Both the abbreviated form `jdbc/oracle` form, as in Tomcat, and the canonical form `java:comp/env/jdbc/oracle` are supported.

This example uses JNDI to obtain Oracle connections:

```
<DataStore>
    <id>datastore</id>
    <parameters>
        <Parameter>
            <name>dbtype</name>
            <value>Oracle</value>
        </Parameter>
        <Parameter>
            <name>jndiReferenceName</name>
            <value>jdbc/oracle</value>
        </Parameter>
    </parameters>
</DataStore>
```

Your servlet container my require you to add a `resource-ref` section at the end of your `geoserver/WEB-INF/web.xml`. (Tomcat requires this, Jetty does not.) For example:

```
<resource-ref>
    <description>Oracle Spatial Datasource</description>
    <res-ref-name>jdbc/oracle</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
</resource-ref>
```

Here is an example of a Tomcat 6 context in `/etc/tomcat6/server.xml` that includes an Oracle connection pool:

```
<Context
    path="/geoserver"
    docBase="/usr/local/geoserver"
    crossContext="false"
    reloadable="false">
    <Resource
        name="jdbc/oracle"
        auth="Container"
        type="javax.sql.DataSource"
        url="jdbc:oracle:thin:@YOUR_DATABASE_HOSTNAME:1521:YOUR_DATABASE_NAME"
        driverClassName="oracle.jdbc.driver.OracleDriver"
        username="YOUR_DATABASE_USERNAME"
        password="YOUR_DATABASE_PASSWORD"
        maxActive="20"
        maxIdle="10"
        minIdle="0"
        maxWait="10000"
        minEvictableIdleTimeMillis="300000"
        timeBetweenEvictionRunsMillis="300000"
        numTestsPerEvictionRun="20"
        poolPreparedStatements="true"
        maxOpenPreparedStatements="100"
        testOnBorrow="true"
        validationQuery="SELECT SYSDATE FROM DUAL" />
</Context>
```

Firewall timeouts can silently sever idle connections to the database and cause GeoServer to hang. If there is a firewall between GeoServer and the database, a connection pool configured to shut down idle connections before the firewall can drop them will prevent GeoServer from hanging. This JNDI connection pool is

configured to shut down idle connections after 5 to 10 minutes.

See also *Setting up a JNDI connection pool with Tomcat*.

Expose primary keys

By default, GeoServer conceals the existence of database columns with a primary key. To make such columns available for use in app-schema mapping files, set the data store parameter `Expose primary keys` to `true`:

```
<Parameter>
    <name>Expose primary keys</name>
   <value>true</value>
</Parameter>
```

This is known to work with PostGIS, Oracle, and JNDI data stores.

MongoDB

The data store configuration for a MongoDB data base will look like this:

```
<sourceDataStores>
    <DataStore>
        <id>data_source</id>
        <parameters>
            <Parameter>
                <name>data_store</name>
                <value>MONGO_DB_URL</value>
            </Parameter>
            <Parameter>
                <name>namespace</name>
                <value>NAME_SPACE</value>
            </Parameter>
            <Parameter>
                <name>schema_store</name>
                <value>SCHEMA_STORE</value>
            </Parameter>
            <Parameter>
                <name>data_store_type</name>
                <value>complex</value>
            </Parameter>
        </parameters>
    </DataStore>
</sourceDataStores>
```

Check *MongoDB Tutorial* for a more detailed description about how to use MongoDB with app-schema.

Note: You must install the MongoDB plugin to connect to MongoDB databases.

5.6.12 Feature Chaining

Scope

This page describes the use of "Feature Chaining" to compose complex features from simpler components, and in particular to address some requirements that have proven significant in practice.

- Handling multiple cases of multi-valued properties within a single Feature Type
- Handing nesting of multi-valued properties within other multi-valued properties
- Linking related (through association) Feature Types, and in particular allowing re-use of the related features types (for example the O&M pattern has relatedObservation from a samplingFeature, but Observation may be useful in its own right)
- Encoding the same referenced property object as links when it appears in multiple containing features
- Eliminating the need for large denormalized data store views of top level features and their related features. Denormalized views would still be needed for special cases, such as many-to-many relationships, but won't be as large.

For non-application schema configurations, please refer to *Data Access Integration*.

Mapping steps

Create a mapping file for every complex type

We need one mapping file per complex type that is going to be nested, including non features, e.g. gsml:CompositionPart.

Non-feature types that cannot be individually accessed (eg. CompositionPart as a Data Type) can still be mapped separately for its reusability. For this case, the containing feature type has to include these types in its mapping file. The include tag should contain the nested mapping file path relative to the location of the containing type mapping file. In `GeologicUnit_MappingFile.xml`:

```
<includedTypes>
    <Include>CGITermValue_MappingFile.xml</Include>
    <Include>CompositionPart_MappingFile.xml</Include>
</includedTypes>
```

Feature types that can be individually accessed don't need to be explicitly included in the mapping file, as they would be configured for GeoServer to find. Such types would have their mapping file associated with a corresponding datastore.xml file, which means that it can be found from the data store registry. In other words, if the type is associated with a datastore.xml file, it doesn't need to be explicitly included if referred from another mapping file.

Example:

For this output: `MappedFeature_Output.xml`, here are the mapping files:

- `MappedFeature_MappingFile.xml`
- `GeologicUnit_MappingFile.xml`
- `CompositionPart_MappingFile.xml`
- `GeologicEvent_MappingFile.xml`
- `CGITermValue_MappingFile.xml`

GeologicUnit type

You can see within GeologicUnit features, both gml:composition (CompositionPart type) and gsml:geologicHistory (GeologicEvent type) are multi-valued properties. It shows how multiple cases of multi-valued properties can be configured within a single Feature Type. This also proves that you can "chain" non-feature type, as CompositionPart is a Data Type.

GeologicEvent type

Both gsml:eventEnvironment (CGI_TermValue type) and gsml:eventProcess (also of CGI_TermValue type) are multi-valued properties. This also shows that "chaining" can be done on many levels, as GeologicEvent is nested inside GeologicUnit. Note that gsml:eventAge properties are configured as inline attributes, as there can only be one event age per geologic event, thus eliminating the need for feature chaining.

Configure nesting on the nested feature type

In the nested feature type, make sure we have a field that can be referenced by the parent feature. If there isn't any existing field that can be referred to, the system field *FEATURE_LINK* can be mapped to hold the foreign key value. This is a multi-valued field, so more than one instances can be mapped in the same feature type, for features that can be nested by different parent types. Since this field doesn't exist in the schema, it wouldn't appear in the output document.

In the source expression tag:

- OCQL: the value of this should correspond to the OCQL part of the parent feature

Example One: Using *FEATURE_LINK* in CGI TermValue type, which is referred by GeologicEvent as gsml:eventProcess and gsml:eventEnvironment.

In GeologicEvent (the container feature) mapping:

```
<AttributeMapping>
      <targetAttribute>gsml:eventEnvironment</targetAttribute>
      <sourceExpression>
              <OCQL>id</OCQL>
              <linkElement>gsml:CGI_TermValue</linkElement>
              <linkField>FEATURE_LINK[1]</linkField>
      </sourceExpression>
      <isMultiple>true</isMultiple>
</AttributeMapping>
<AttributeMapping>
      <targetAttribute>gsml:eventProcess</targetAttribute>
      <sourceExpression>
              <OCQL>id</OCQL>
              <linkElement>gsml:CGI_TermValue</linkElement>
              <linkField>FEATURE_LINK[2]</linkField>
      </sourceExpression>
      <isMultiple>true</isMultiple>
</AttributeMapping>
```

In CGI_TermValue (the nested feature) mapping:

```
<AttributeMapping>
  <!-- FEATURE_LINK[1] is referred by geologic event as environment -->
  <targetAttribute>FEATURE_LINK[1]</targetAttribute>
  <sourceExpression>
      <OCQL>ENVIRONMENT_OWNER</OCQL>
  </sourceExpression>
</AttributeMapping>
```

```
<AttributeMapping>
  <!-- FEATURE_LINK[2] is referred by geologic event as process -->
  <targetAttribute>FEATURE_LINK[2]</targetAttribute>
  <sourceExpression><
      <OCQL>PROCESS_OWNER</OCQL>
  </sourceExpression>
</AttributeMapping>
```

The ENVIRONMENT_OWNER column in CGI_TermValue view corresponds to the ID column in GeologicEvent view.

Geologic Event property file:

id	GEOLOGIC_gmin:String	gmax:String	gmaxage:String	gmin_cdspace:String
ge.26931120	gu.25699	Oligocene	Paleocene	urn:cgi:classifierScheme:ICS:StratChart:2008
ge.26930473	gu.25678	Holocene	Pleistocene	urn:cgi:classifierScheme:ICS:StratChart:2008
ge.26930960	gu.25678	Pliocene	Miocene	urn:cgi:classifierScheme:ICS:StratChart:2008
ge.26932959	gu.25678	LowerOrdovician	LowerOrdovician	urn:cgi:classifierScheme:ICS:StratChart:2008

CGI Term Value property file:

id	VALUE:String	PROCESS_OWNER:String	ENVIRONMENT_OWNER:String
3	fluvial	NULL	ge.26931120
4	swamp/marsh/bog	NULL	ge.26930473
5	marine	NULL	ge.26930960
6	submarine fan	NULL	ge.26932959
7	hemipelagic	NULL	ge.26932959
8	detrital deposition still water	ge.26930473	NULL
9	water [process]	ge.26932959	NULL
10	channelled stream flow	ge.26931120	NULL
11	turbidity current	ge.26932959	NULL

The system field *FEATURE_LINK* doesn't get encoded in the output:

```
<gsml:GeologicEvent>
  <gml:name codeSpace="urn:cgi:classifierScheme:GSV:GeologicalUnitId">gu.25699</
↪gml:name>
  <gsml:eventAge>
    <gsml:CGI_TermRange>
       <gsml:lower>
          <gsml:CGI_TermValue>
            <gsml:value codeSpace="urn:cgi:classifierScheme:ICS:StratChart:2008">
↪Oligocene</gsml:value>
          </gsml:CGI_TermValue>
       </gsml:lower>
       <gsml:upper>
          <gsml:CGI_TermValue>
            <gsml:value codeSpace="urn:cgi:classifierScheme:ICS:StratChart:2008">
↪Paleocene</gsml:value>
          </gsml:CGI_TermValue>
       </gsml:upper>
    </gsml:CGI_TermRange>
  </gsml:eventAge>
  <gsml:eventEnvironment>
    <gsml:CGI_TermValue>
```

```
        <gsml:value>fluvial</gsml:value>
      </gsml:CGI_TermValue>
    </gsml:eventEnvironment>
    <gsml:eventProcess>
      <gsml:CGI_TermValue>
         <gsml:value>channelled stream flow</gsml:value>
      </gsml:CGI_TermValue>
    </gsml:eventProcess>
```

Example Two: Using existing field (gml:name) to hold the foreign key, see `MappedFeature_MappingFile.xml`:

gsml:specification links to gml:name in GeologicUnit:

```
<AttributeMapping>
  <targetAttribute>gsml:specification</targetAttribute>
  <sourceExpression>
    <OCQL>GEOLOGIC_UNIT_ID</OCQL>
    <linkElement>gsml:GeologicUnit</linkElement>
    <linkField>gml:name[3]</linkField>
  </sourceExpression>
</AttributeMapping>
```

In `GeologicUnit_MappingFile.xml`:

GeologicUnit has 3 gml:name properties in the mapping file, so each has a code space to clarify them:

```
<AttributeMapping>
  <targetAttribute>gml:name[1]</targetAttribute>
  <sourceExpression>
    <OCQL>ABBREVIATION</OCQL>
  </sourceExpression>
  <ClientProperty>
    <name>codeSpace</name>
    <value>'urn:cgi:classifierScheme:GSV:GeologicalUnitCode'</value>
  </ClientProperty>
</AttributeMapping>
<AttributeMapping>
  <targetAttribute>gml:name[2]</targetAttribute>
  <sourceExpression>
    <OCQL>NAME</OCQL>
  </sourceExpression>
  <ClientProperty>
    <name>codeSpace</name>
    <value>'urn:cgi:classifierScheme:GSV:GeologicalUnitName'</value>
  </ClientProperty>
</AttributeMapping>
<AttributeMapping>
  <targetAttribute>gml:name[3]</targetAttribute>
  <sourceExpression>
    <OCQL>id</OCQL>
  </sourceExpression>
  <ClientProperty>
    <name>codeSpace</name>
    <value>'urn:cgi:classifierScheme:GSV:MappedFeatureReference'</value>
  </ClientProperty>
</AttributeMapping>
```

The output with multiple gml:name properties and their code spaces:

```
<gsml:specification>
  <gsml:GeologicUnit gml:id="gu.25678">
      <gml:description>Olivine basalt, tuff, microgabbro, minor sedimentary rocks</
↪gml:description>
      <gml:name codeSpace="urn:cgi:classifierScheme:GSV:GeologicalUnitCode">-Py</
↪gml:name>
      <gml:name codeSpace="urn:cgi:classifierScheme:GSV:GeologicalUnitName">Yaugher␣
↪Volcanic Group</gml:name>
      <gml:name codeSpace="urn:cgi:classifierScheme:GSV:MappedFeatureReference">gu.
↪25678</gml:name>
```

If this is the "one" side of a one-to-many or many-to-one database relationship, we can use the feature id as the source expression field, as you can see in above examples. See `one_to_many_relationship.JPG` as an illustration.

If we have a many-to-many relationship, we have to use one denormalized view for either side of the nesting. This means we can either use the feature id as the referenced field, or assign a column to serve this purpose. See `many_to_many_relationship.JPG` as an illustration.

Note:

- For many-to-many relationships, we can't use the same denormalized view for both sides of the nesting.

Test this configuration by running a getFeature request for the nested feature type on its own.

Configure nesting on the "containing" feature type

When nesting another complex type, you need to specify in your source expression:

- **OCQL**: OGC's Common Query Language expression of the data store column
- **linkElement:**
  - the nested element name, which is normally the targetElement or mappingName of the corresponding type.
  - on some cases, it has to be an OCQL function (see *Polymorphism*)
- **linkField**: the indexed XPath attribute on the nested element that OCQL corresponds to

Example: Nesting composition part in geologic unit feature.

In Geologic Unit mapping file:

```
<AttributeMapping>
    <targetAttribute>gsml:composition</targetAttribute>
    <sourceExpression>
            <OCQL>id</OCQL>
            <linkElement>gsml:CompositionPart</linkElement>
            <linkField>FEATURE_LINK</linkField>
    </sourceExpression>
    <isMultiple>true</isMultiple>
</AttributeMapping>
```

- *OCQL*: id is the geologic unit id
- *linkElement*: links to gsml:CompositionPart type

- *linkField*: FEATURE_LINK, the linking field mapped in gsml:CompositionPart type that also stores the geologic unit id. If there are more than one of these attributes in the nested feature type, make sure the index is included, e.g. FEATURE_LINK[2].

Geologic Unit property file:

id	ABBREVIATION:String	NAME:String	TEXTDESCRIPTION:String
gu.25699	-Py	Yaugher Volcanic Group	Olivine basalt, tuff, microgabbro, minor sedimentary rocks
gu.25678	-Py	Yaugher Volcanic Group	Olivine basalt, tuff, microgabbro, minor sedimentary rocks

Composition Part property file:

id	COMPONENT_ROLE:String	PROPORTION:String	GEOLOGIC_UNIT_ID:String
cp.167775491936278812	interbedded component	significant	gu.25699
cp.167775491936278856	interbedded component	minor	gu.25678
cp.167775491936278844	sole component	major	gu.25678

Run the getFeature request to test this configuration. Check that the nested features returned in Step 2 are appropriately lined inside the containing features. If they are not there, or exceptions are thrown, scroll down and read the "Trouble Shooting" section.

Multiple mappings of the same type

At times, you may find the need to have different FeatureTypeMapping instances for the same type. You may have two different attributes of the same type that need to be nested. For example, in gsml:GeologicUnit, you have gsml:exposureColor and gsml:outcropCharacter that are both of gsml:CGI_TermValue type.

This is when the optional mappingName tag mentioned in *Mapping File* comes in. Instead of passing in the nested feature type's targetElement in the containing type's linkElement, specify the corresponding mappingName.

Note:

- The mappingName is namespace aware and case sensitive.
- When the referred mappingName contains special characters such as '-', it must be enclosed with single quotes in the linkElement. E.g. <linkElement>'observation-method'</linkElement>.
- Each mappingName must be unique against other mappingName and targetElement tags across the application.
- The mappingName is only to be used to identify the chained type from the nesting type. It is not a solution for multiple FeatureTypeMapping instances where > 1 of them can be queried as top level features.
- When queried as a top level feature, the normal targetElement is to be used. Filters involving the nested type should still use the targetElement in the PropertyName part of the query.
- You can't have more than 1 FeatureTypeMapping of the same type in the same mapping file if one of them is a top level feature. This is because featuretype.xml would look for the targetElement and wouldn't know which one to get.

The solution for the last point above is to break them up into separate files and locations with only 1 featuretype.xml in the intended top level feature location. E.g.

- You can have 2 FeatureTypeMapping instances in the same file for gsml:CGI_TermValue type since it's not a feature type.
- You can have 2 FeatureTypeMapping instances for gsml:MappedFeature, but they have to be broken up into separate files. The one that can be queried as top level feature type would have featuretype.xml in its location.

Nesting simple properties

You don't need to chain multi-valued simple properties and map them separately. The original configuration would still work.

Filtering nested attributes on chained features

Filters would work as usual. You can supply the full XPath of the attribute, and the code would handle this. E.g. You can run the following filter on gsml:MappedFeatureUseCase2A:

```
<ogc:Filter>
      <ogc:PropertyIsEqualTo>
          <ogc:Function name="contains_text">
              <ogc:PropertyName>gsml:specification/gsml:GeologicUnit/gml:description</
↪ogc:PropertyName>
              <ogc:Literal>Olivine basalt, tuff, microgabbro, minor sedimentary rocks
↪</ogc:Literal>
          </ogc:Function>
          <ogc:Literal>1</ogc:Literal>
      </ogc:PropertyIsEqualTo>
</ogc:Filter>
```

Multi-valued properties by reference (*xlink:href*)

You may want to use feature chaining to set multi-valued properties by reference. This is particularly handy to avoid endless loop in circular relationships. For example, you may have a circular relationship between gsml:MappedFeature and gsml:GeologicUnit. E.g.

- gsml:MappedFeature has gsml:GeologicUnit as gsml:specification
- gsml:GeologicUnit has gsml:MappedFeature as gsml:occurrence

Obviously you can only encode one side of the relationship, or you'll end up with an endless loop. You would need to pick one side to "chain" and use xlink:href for the other side of the relationship.

For this example, we are nesting gsml:GeologicUnit in gsml:MappedFeature as gsml:specification.

- Set up nesting on the container feature type mapping as usual:

```
<AttributeMapping>
  <targetAttribute>gsml:specification</targetAttribute>
  <sourceExpression>
      <OCQL>GEOLOGIC_UNIT_ID</OCQL>
        <linkElement>gsml:GeologicUnit</linkElement>
        <linkField>gml:name[2]</linkField>
```

```
  </sourceExpression>
</AttributeMapping>
```

- Set up xlink:href as client property on the other mapping file:

```
<AttributeMapping>
  <targetAttribute>gsml:occurrence</targetAttribute>
  <sourceExpression>
        <OCQL>id</OCQL>
        <linkElement>gsml:MappedFeature</linkElement>
        <linkField>gsml:specification</linkField>
  </sourceExpression>
  <isMultiple>true</isMultiple>
  <ClientProperty>
         <name>xlink:href</name>
         <value>strConcat('urn:cgi:feature:MappedFeature:', ID)</value>
  </ClientProperty>
</AttributeMapping>
```

As we are getting the client property value from a nested feature, we have to set it as if we are chaining the feature; but we also add the client property containing *xlink:href* in the attribute mapping. The code will detect the *xlink:href* setting, and will not proceed to build the nested feature's attributes, and we will end up with empty attributes with *xlink:href* client properties.

This would be the encoded result for gsml:GeologicUnit:

```
<gsml:GeologicUnit gml:id="gu.25678">
         <gsml:occurrence xlink:href="urn:cgi:feature:MappedFeature:mf2"/>
         <gsml:occurrence xlink:href="urn:cgi:feature:MappedFeature:mf3"/>
```

Note:

- Don't forget to add *XLink* in your mapping file namespaces section, or you could end up with a StackOverflowException as the *xlink:href* client property won't be recognized and the mappings would chain endlessly.
- *Resolving* may be used to force app-schema to do full feature chaining up to a certain level, even if an xlink reference is specified.

5.6.13 Polymorphism

Polymorphism in this context refers to the ability of an attribute to have different forms. Depending on the source value, it could be encoded with a specific structure, type, as an xlink:href reference, or not encoded at all. To achieve this, we reuse feature chaining syntax and allow OCQL functions in the linkElement tag. Read more about *Feature Chaining*, if you're not familiar with the syntax.

Data-type polymorphism

You can use normal feature chaining to get an attribute to be encoded as a certain type. For example:

```
<AttributeMapping>
    <targetAttribute>ex:someAttribute</targetAttribute>
    <sourceExpression>
```

```
        <OCQL>VALUE_ID</OCQL>
          <linkElement>NumericType</linkElement>
          <linkField>FEATURE_LINK</linkField>
    </sourceExpression>
</AttributeMapping>
<AttributeMapping>
    <targetAttribute>ex:someAttribute</targetAttribute>
    <sourceExpression>
          <OCQL>VALUE_ID</OCQL>
          <linkElement>gsml:CGI_TermValue</linkElement>
          <linkField>FEATURE_LINK</linkField>
    </sourceExpression>
</AttributeMapping>
```

Note: NumericType here is a mappingName, whereas gsml:CGI_TermValue is a targetElement.

In the above example, ex:someAttribute would be encoded with the configuration in NumericType if the foreign key matches the linkField. Both instances would be encoded if the foreign key matches the candidate keys in both linked configurations. Therefore this would only work for 0 to many relationships.

Functions can be used for single attribute instances. See *useful functions* for a list of commonly used functions. Specify the function in the linkElement, and it would map it to the first matching FeatureTypeMapping. For example:

```
<AttributeMapping>
    <targetAttribute>ex:someAttribute</targetAttribute>
    <sourceExpression>
          <OCQL>VALUE_ID</OCQL>
          <linkElement>
            Recode(CLASS_TEXT, 'numeric', 'NumericType', 'literal', 'gsml:CGI_
↪TermValue')
        </linkElement>
          <linkField>FEATURE_LINK</linkField>
    </sourceExpression>
    <isMultiple>true</isMultiple>
</AttributeMapping>
```

The above example means, if the CLASS_TEXT value is 'numeric', it would link to 'NumericType' FeatureTypeMapping, with VALUE_ID as foreign key to the linked type. It would require all the potential matching types to have a common attribute that is specified in linkField. In this example, the linkField is FEATURE_LINK, which is a fake attribute used only for feature chaining. You can omit the linkField and OCQL if the FeatureTypeMapping being linked to has the same sourceType with the container type. This would save us from unnecessary extra queries, which would affect performance. For example:

FeatureTypeMapping of the container type:

```
<FeatureTypeMapping>
    <sourceDataStore>PropertyFiles</sourceDataStore>
    <sourceType>PolymorphicFeature</sourceType>
```

FeatureTypeMapping of NumericType points to the same table:

```
<FeatureTypeMapping>
    <mappingName>NumericType</mappingName>
    <sourceDataStore>PropertyFiles</sourceDataStore>
    <sourceType>PolymorphicFeature</sourceType>
```

FeatureTypeMapping of gsml:CGI_TermValue also points to the same table:

```
<FeatureTypeMapping>
    <sourceDataStore>PropertyFiles</sourceDataStore>
    <sourceType>PolymorphicFeature</sourceType>
    <targetElement>gsml:CGI_TermValue</targetElement>
```

In this case, we can omit linkField in the polymorphic attribute mapping:

```
<AttributeMapping>
    <targetAttribute>ex:someAttribute</targetAttribute>
    <sourceExpression>
          <linkElement>
            Recode(CLASS_TEXT, 'numeric', 'NumericType', 'literal', 'gsml:CGI_
↪TermValue')
        </linkElement>
    </sourceExpression>
    <isMultiple>true</isMultiple>
</AttributeMapping>
```

Referential polymorphism

This is when an attribute is set to be encoded as an xlink:href reference on the top level. When the scenario only has reference cases in it, setting a function in Client Property will do the job. E.g.:

```
<AttributeMapping>
      <targetAttribute>ex:someAttribute</targetAttribute>
      <ClientProperty>
            <name>xlink:href</name>
            <value>if_then_else(isNull(NUMERIC_VALUE), 'urn:ogc:def:nil:OGC:1.
↪0:missing', strConcat('#', NUMERIC_VALUE))</value>
    </ClientProperty>
</AttributeMapping>
```

The above example means, if NUMERIC_VALUE is null, the attribute should be encoded as:

```
<ex:someAttribute xlink:href="urn:ogc:def:nil:OGC:1.0:missing">
```

Otherwise, it would be encoded as:

```
<ex:someAttribute xlink:href="#123">
    where NUMERIC_VALUE = '123'
```

However, this is not possible when we have cases where a fully structured attribute is also a possibility. The *toxlinkhref* function can be used for this scenario. E.g.:

```
<AttributeMapping>
    <targetAttribute>ex:someAttribute</targetAttribute>
    <sourceExpression>
          <linkElement>
            if_then_else(isNull(NUMERIC_VALUE), toXlinkHref('urn:ogc:def:nil:OGC:1.
↪0:missing'),
              if_then_else(lessEqualThan(NUMERIC_VALUE, 1000), 'numeric_value',␣
↪toXlinkHref('urn:ogc:def:nil:OGC:1.0:missing')))
        </linkElement>
    </sourceExpression>
</AttributeMapping>
```

The above example means, if NUMERIC_VALUE is null, the output would be encoded as:

```
<ex:someAttribute xlink:href="urn:ogc:def:nil:OGC:1.0:missing">
```

Otherwise, if NUMERIC_VALUE is less or equal than 1000, it would be encoded with attributes from FeatureTypeMapping with 'numeric_value' mappingName. If NUMERIC_VALUE is greater than 1000, it would be encoded as the first scenario.

Useful functions

if_then_else function

Syntax:

```
if_then_else(BOOLEAN_EXPRESSION, value, default value)
```

- **BOOLEAN_EXPRESSION**: could be a Boolean column value, or a Boolean function
- **value**: the value to map to, if BOOLEAN_EXPRESSION is true
- **default value**: the value to map to, if BOOLEAN_EXPRESSION is false

Recode function

Syntax:

```
Recode(EXPRESSION, key1, value1, key2, value2,...)
```

- **EXPRESSION**: column name to get values from, or another function
- **key-n:**
  - key expression to map to value-n
  - if the evaluated value of EXPRESSION doesn't match any key, nothing would be encoded for the attribute.
- **value-n**: value expression which translates to a mappingName or targetElement

lessEqualThan

Returns true if ATTRIBUTE_EXPRESSION evaluates to less or equal than LIMIT_EXPRESSION.

Syntax:

```
lessEqualThan(ATTRIBUTE_EXPRESSION, LIMIT_EXPRESSION)
```

- **ATTRIBUTE_EXPRESSION**: expression of the attribute being evaluated.
- **LIMIT_EXPRESSION**: expression of the numeric value to be compared against.

lessThan

Returns true if ATTRIBUTE_EXPRESSION evaluates to less than LIMIT_EXPRESSION.

Syntax:

```
lessThan(ATTRIBUTE_EXPRESSION, LIMIT_EXPRESSION)
```

- **ATTRIBUTE_EXPRESSION**: expression of the attribute being evaluated.
- **LIMIT_EXPRESSION**: expression of the numeric value to be compared against.

equalTo

Compares two expressions and returns true if they're equal.

Syntax:

```
equalTo(LHS_EXPRESSION, RHS_EXPRESSION)
```

isNull

Returns a Boolean that is true if the expression evaluates to null.

Syntax:

```
isNull(EXPRESSION)
```

- **EXPRESSION**: expression to be evaluated.

toXlinkHref

Special function written for referential polymorphism and feature chaining, not to be used outside of linkElement. It infers that the attribute should be encoded as xlink:href.

Syntax:

```
toXlinkHref(XLINK_HREF_EXPRESSION)
```

- **XLINK_HREF_EXPRESSION:**
  - could be a function or a literal
  - has to be wrapped in single quotes if it's a literal

Note:

- To get toXlinkHref function working, you need to declare xlink URI in the namespaces.

Other functions

Please refer to *Filter Function Reference*.

Combinations

You can combine functions, but it might affect performance. E.g.:

```
if_then_else(isNull(NUMERIC_VALUE), toXlinkHref('urn:ogc:def:nil:OGC:1.0:missing'),
    if_then_else(lessEqualThan(NUMERIC_VALUE, 1000), 'numeric_value', toXlinkHref(
↪'urn:ogc:def:nil:OGC:1.0:missing')))
```

Note:

- When specifying a mappingName or targetElement as a value in functions, make sure they're enclosed in single quotes.
- Some functions have no null checking, and will fail when they encounter null.
- The workaround for this is to wrap the expression with isNull() function if null is known to exist in the data set.

Null or missing value

To skip the attribute for a specific case, you can use Expression.NIL as a value in if_then_else or not include the key in *Recode function* . E.g.:

```
if_then_else(isNull(VALUE), Expression.NIL, 'gsml:CGI_TermValue')
    means the attribute would not be encoded if VALUE is null.

Recode(VALUE, 'term_value', 'gsml:CGI_TermValue')
    means the attribute would not be encoded if VALUE is anything but 'term_value'.
```

To encode an attribute as xlink:href that represents missing value on the top level, see *Referential Polymorphism*.

Any type

Having xs:anyType as the attribute type itself infers that it is polymorphic, since they can be encoded as any type.

If the type is pre-determined and would always be the same, we might need to specify *targetAttributeNode (optional)*. E.g.:

```
<AttributeMapping>
      <targetAttribute>om:result</targetAttribute>
      <targetAttributeNode>gml:MeasureType<targetAttributeNode>
      <sourceExpression>
          <OCQL>TOPAGE</OCQL>
      </sourceExpression>
      <ClientProperty>
          <name>xsi:type</name>
          <value>'gml:MeasureType'</value>
      </ClientProperty>
      <ClientProperty>
          <name>uom</name>
          <value>'http://www.opengis.net/def/uom/UCUM/0/Ma'</value>
      </ClientProperty>
</AttributeMapping>
```

If the casting type is complex, this is not a requirement as app-schema is able to automatically determine the type from the XPath in targetAttribute. E.g., in this example `om:result` is automatically specialised as a MappedFeatureType:

```
<AttributeMapping>
      <targetAttribute>om:result/gsml:MappedFeature/gml:name</targetAttribute>
      <sourceExpression>
          <OCQL>NAME</OCQL>
      </sourceExpression>
</AttributeMapping>
```

Alternatively, we can use feature chaining. For the same example above, the mapping would be:

```
<AttributeMapping>
      <targetAttribute>om:result</targetAttribute>
      <sourceExpression>
            <OCQL>LEX_D</OCQL>
            <linkElement>gsml:MappedFeature</linkElement>
            <linkField>gml:name</linkField>
      </sourceExpression>
</AttributeMapping>
```

If the type is conditional, the mapping style for such attributes is the same as any other polymorphic attributes. E.g.:

```
<AttributeMapping>
      <targetAttribute>om:result</targetAttribute>
      <sourceExpression>
            <linkElement>
               Recode(NAME, Expression.Nil, toXlinkHref('urn:ogc:def:nil:OGC::missing
↪'),'numeric',
               toXlinkHref(strConcat('urn:numeric-value::', NUMERIC_VALUE)), 'literal
↪', 'TermValue2')
            </linkElement>
      </sourceExpression>
</AttributeMapping>
```

Filters

Filters should work as usual, as long as the users know what they want to filter. For example, when an attribute could be encoded as gsml:CGI_TermValue or gsml:CGI_NumericValue, users can run filters with property names of:

- ex:someAttribute/gsml:CGI_TermValue/gsml:value to return matching attributes that are encoded as gsml:CGI_TermValue and satisfy the filter.
- likewise, ex:someAttribute/gsml:CGI_NumericValue/gsml:principalValue should return matching gsml:CGI_NumericValue attributes.

Another limitation is filtering attributes of an xlink:href attribute pointing to an instance outside of the document.

5.6.14 Data Access Integration

This page assumes prior knowledge of *Application schemas* and *Feature Chaining*. To use feature chaining, the nested features can come from any complex feature data access, as long as:

- it has valid data referred by the "container" feature type,
- the data access is registered via DataAccessRegistry,
- if FEATURE_LINK is used as the link field, the feature types were created via ComplexFeatureTypeFactoryImpl

However, the "container" features must come from an application schema data access. The rest of this article describes how we can create an application data access from an existing non-application schema data access, in order to "chain" features. The input data access referred in this article is assumed to be the non-application schema data access.

How to connect to the input data access

Configure the data store connection in "sourceDataStores" tag as usual, but also specify the additional "isDataAccess" tag. This flag marks that we want to get the registered complex feature source of the specified "sourceType", when processing the source data store. This assumes that the input data access is registered in DataAccessRegistry upon creation, for the system to find it.

Example:

```
<sourceDataStores>
  <DataStore>
      <id>EarthResource</id>
      <parameters>
         <Parameter>
           <name>directory</name>
           <value>file:./</value>
         </Parameter>
      </parameters>
      <isDataAccess>true</isDataAccess>
  </DataStore>
</sourceDataStores>
...
<typeMappings>
  <FeatureTypeMapping>
    <sourceDataStore>EarthResource</sourceDataStore>
      <sourceType>EarthResource</sourceType>
...
```

How to configure the mapping

Use "inputAttribute" in place of "OCQL" tag inside "sourceExpression", to specify the input XPath expressions.

Example:

```
<AttributeMapping>
  <targetAttribute>gsml:classifier/gsml:ControlledConcept/gsml:preferredName</
↪targetAttribute>
  <sourceExpression>
      <inputAttribute>mo:classification/mo:MineralDepositModel/mo:mineralDepositGroup
↪</inputAttribute>
```

```
  </sourceExpression>
</AttributeMapping>
```

How to chain features

Feature chaining works both ways for the re-mapped complex features. You can chain other features inside these features, and vice-versa. The only difference is to use "inputAttribute" for the input XPath expressions, instead of "OCQL" as mentioned above.

Example:

```
<AttributeMapping>
  <targetAttribute>gsml:occurence</targetAttribute>
  <sourceExpression>
      <inputAttribute>mo:commodityDescription</inputAttribute>
      <linkElement>gsml:MappedFeature</linkElement>
      <linkField>gml:name[2]</linkField>
  </sourceExpression>
  <isMultiple>true</isMultiple>
</AttributeMapping>
```

How to use filters

From the user point of view, filters are configured as per normal, using the mapped/output target attribute XPath expressions. However, when one or more attributes in the expression is a multi-valued property, we need to specify a function such as "contains_text" in the filter. This is because when multiple values are returned, comparing them to a single value would only return true if there is only one value returned, and it is the same value. Please note that the "contains_text" function used in the following example is not available in GeoServer API, but defined in the database.

Example:

Composition is a multi-valued property:

```
<ogc:Filter>
  <ogc:PropertyIsEqualTo>
    <ogc:Function name="contains_text">
        <ogc:PropertyName>gsml:composition/gsml:CompositionPart/gsml:proportion/
↪gsml:CGI_TermValue/gsml:value</ogc:PropertyName>
        <ogc:Literal>Olivine basalt, tuff, microgabbro, minor sedimentary rocks</
↪ogc:Literal>
    </ogc:Function>
    <ogc:Literal>1</ogc:Literal>
  </ogc:PropertyIsEqualTo>
</ogc:Filter>
```

5.6.15 WMS Support

App-schema supports WMS requests as well as WFS requests. This page provides some useful examples for configuring the WMS service to work with complex features.

Note that the rendering performance of WMS can be significantly slower when using app-schema data stores. We strongly recommend employing *Joining Support For Performance* when using WMS with feature chaining, which can make response time for large data requests several orders of magnitude faster.

Configuration

For WMS to be applicable to complex feature data, it is necessary that the complex feature types are recognised by GeoServer as layers. This must be configured by adding an extra configuration file named 'layer.xml' in the data directory of each feature type that we want to use as a WMS layer.

This will expand the structure of the `workspaces` folder in the GeoServer data directory as follows (`workspaces`) (see *Configuration*):

```
workspaces
    - gsml
        - SomeDataStore
            - SomeFeatureType
                - featuretype.xml
                - layer.xml
            - datastore.xml
            - SomeFeatureType-mapping-file.xml
```

The file layer.xml must have the following contents:

```
<layer>
  <id>[mylayerid]</id>
  <name>[mylayername]</name>
  <path>/</path>
  <type>VECTOR</type>
  <defaultStyle>
        <name>[mydefaultstyle]</name>
  </defaultStyle>
  <resource class="featureType">
        <id>[myfeaturetypeid]</id>
  </resource>
  <enabled>true</enabled>
  <attribution>
        <logoWidth>0</logoWidth>
        <logoHeight>0</logoHeight>
  </attribution>
</layer>
```

Replace the fields in between brackets with the following values:

- **[mylayerid]** must be a custom id for the layer.
- **[mylayername]** must be a custom name for the layer.
- **[mydefaultstyle]** the default style used for this layer (when a style is not specified in the wms request). The style must exist in the GeoServer configuration.
- **[myfeaturetypeid]** is the id of the feature type. This *must* the same as the id specified in the file featuretype.xml of the same directory.

GetMap

Read *GetMap* for general information on the GetMap request. Read *Styling* for general information on how to style WMS maps with SLD files. When styling complex features, you can use XPaths to specify nested properties in your filters, as explained in *Filtering nested attributes on chained features*. However, in WMS styling filters X-paths do not support handling referenced features (see *Multi-valued properties by reference (xlink:href)*) as if they were actual nested features (because the filters are applied after building the features

rather than before.) The prefix/namespace context that is used in the XPath expression is defined locally in the XML tags of the style file. This is an example of a Style file for complex features:

```
<?xml version="1.0" encoding="UTF-8"?>
<StyledLayerDescriptor version="1.0.0"
    xsi:schemaLocation="http://www.opengis.net/sld StyledLayerDescriptor.xsd"
    xmlns:ogc="http://www.opengis.net/ogc"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:gml="http://www.opengis.net/gml"
    xmlns:gsml="urn:cgi:xmlns:CGI:GeoSciML:2.0"
    xmlns:sld="http://www.opengis.net/sld"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <sld:NamedLayer>
  <sld:Name>geology-lithology</sld:Name>
  <sld:UserStyle>
    <sld:Name>geology-lithology</sld:Name>
    <sld:Title>Geological Unit Lithology Theme</sld:Title>
    <sld:Abstract>The colour has been creatively adapted from Moyer,Hasting
         and Raines, 2005 (http://pubs.usgs.gov/of/2005/1314/of2005-1314.pdf)
         which provides xls spreadsheets for various color schemes.
         plus some creative entries to fill missing entries.
    </sld:Abstract>
    <sld:IsDefault>1</sld:IsDefault>
    <sld:FeatureTypeStyle>
      <sld:Rule>
        <sld:Name>acidic igneous material</sld:Name>
        <sld:Abstract>Igneous material with more than 63 percent SiO2.
                      (after LeMaitre et al. 2002)
        </sld:Abstract>
        <ogc:Filter>
          <ogc:PropertyIsEqualTo>
            <ogc:PropertyName>gsml:specification/gsml:GeologicUnit/gsml:composition/
                gsml:CompositionPart/gsml:lithology/@xlink:href</ogc:PropertyName>
            <ogc:Literal>urn:cgi:classifier:CGI:SimpleLithology:200811:
                         acidic_igneous_material</ogc:Literal>
          </ogc:PropertyIsEqualTo>
        </ogc:Filter>
        <sld:PolygonSymbolizer>
          <sld:Fill>
            <sld:CssParameter name="fill">#FFCCB3</sld:CssParameter>
          </sld:Fill>
        </sld:PolygonSymbolizer>
      </sld:Rule>
      <sld:Rule>
        <sld:Name>acidic igneous rock</sld:Name>
        <sld:Abstract>Igneous rock with more than 63 percent SiO2.
                    (after LeMaitre et al. 2002)
        </sld:Abstract>
        <ogc:Filter>
          <ogc:PropertyIsEqualTo>
            <ogc:PropertyName>gsml:specification/gsml:GeologicUnit/gsml:composition/
                gsml:CompositionPart/gsml:lithology/@xlink:href</ogc:PropertyName>
            <ogc:Literal>urn:cgi:classifier:CGI:SimpleLithology:200811:
                         acidic_igneous_rock</ogc:Literal>
            </ogc:PropertyIsEqualTo>
        </ogc:Filter>
        <sld:PolygonSymbolizer>
          <sld:Fill>
            <sld:CssParameter name="fill">#FECDB2</sld:CssParameter>
```

```
          </sld:Fill>
        </sld:PolygonSymbolizer>
      </sld:Rule>
      ...
    </sld:FeatureTypeStyle>
  </sld:UserStyle>
 </sld:NamedLayer>
</sld:StyledLayerDescriptor>
```

GetFeatureInfo

Read *GetFeatureInfo* for general information on the GetFeatureInfo request. Read the tutorial on *GetFeatureInfo Templates* for information on how to template the html output. If you want to store a separate standard template for complex feature collections, save it under the filename `complex_content.ftl` in the template directory.

Read the tutorial on *Freemarker Templates* for more information on how to use the freemarker templates. Freemarker templates support recursive calls, which can be useful for templating complex content. For example, the following freemarker template creates a table of features with a column for each property, and will create another table inside each cell that contains a feature as property:

```
<#--
Macro's used for content
-->

<#macro property node>
    <#if !node.isGeometry>
      <#if node.isComplex>
      <td> <@feature node=node.rawValue type=node.type /> </td>
      <#else>
      <td>${node.value?string}</td>
      </#if>
    </#if>
</#macro>

<#macro header typenode>
<caption class="featureInfo">${typenode.name}</caption>
  <tr>
  <th>fid</th>
<#list typenode.attributes as attribute>
  <#if !attribute.isGeometry>
    <#if attribute.prefix == "">
        <th >${attribute.name}</th>
    <#else>
        <th >${attribute.prefix}:${attribute.name}</th>
    </#if>
  </#if>
</#list>
  </tr>
</#macro>

<#macro feature node type>
<table class="featureInfo">
  <@header typenode=type />
  <tr>
  <td>${node.fid}</td>
```

```
  <#list node.attributes as attribute>
      <@property node=attribute />
  </#list>
  </tr>
</table>
</#macro>

<#--
Body section of the GetFeatureInfo template, it's provided with one feature␣
↪collection, and
will be called multiple times if there are various feature collections
-->
<table class="featureInfo">
  <@header typenode=type />

<#assign odd = false>
<#list features as feature>
  <#if odd>
    <tr class="odd">
  <#else>
    <tr>
  </#if>
  <#assign odd = !odd>

  <td>${feature.fid}</td>
  <#list feature.attributes as attribute>
    <@property node=attribute />
  </#list>
  </tr>
</#list>
</table>
<br/>
```

5.6.16 WFS 2.0 Support

Resolving

Local resolve is supported in app-schema. This can be done by setting the 'resolve' parameter to either 'local' or 'all'. (Remote Resolving is not supported.) The parameter 'resolveDepth' specifies how many levels of references will be resolved. The parameter 'resolveTimeOut' may be used to specify, in seconds, an upper limit to how long app-schema should search for the feature needed for resolving. If the time out limit is reached, the feature is not resolved.

When resolving without Feature Chaining (see below), a GML ID is extracted from the x-link reference and a brute force is done on all feature types to find a feature with this GML ID. The extraction of this GML ID from the Xlink Reference is done using the following rules:

- In case of a URN: The GML ID comes after last colon in the URN. Make sure that the *full* GML ID is included after the last colon (including a possible feature type prefix).
- In case of a URL: The GML ID comes after the # symbol.

Failing to respect one of these rules will result in failure of resolve.

Resolving and Feature Chaining By Reference

The 'resolve' and 'resolveDepth' parameters may also be used in the case of *Multi-valued properties by reference (xlink:href)*. In this case, no brute force will take place, but resolving will instruct App-Schema to do full feature chaining rather than inserting a reference. The URI will not be used to find the feature, but the feature chaining parameters specified in the mapping, as with normal feature chaining. Because of this, the parameter 'resolveTimeOut' will be ignored in this case.

However, be aware that every feature can only appear once in a response. If resolving would break this rule, for example with circular references, the encoder will change the resolved feature back to an (internal) x-link reference.

GetPropertyValue

The GetPropertyValue request is now fully supported. Resolving is also possible in this request, following the same rules as described above.

Paging

Paging is now supported in App-Schema. There are a few exceptions:

- Paging is only supported for data stores with JDBC back ends and will not work for data stores with property files. It has been tested with Oracle and PostGIS databases.
- Paging with filters involving attributes that are mapped to functions will not be supported, as this cannot be translated into SQL.

For more efficient SQL queries generation, please set `isDenormalised` to false where applicable (when a one to one database table is used). See *Mapping File*.

5.6.17 Joining Support For Performance

App-schema joining is a optional configuration parameter that tells app-schema to use a different implementation for *Feature Chaining*, which in many cases can improve performance considerably, by reducing the amount of SQL queries sent to the DBMS.

Conditions

In order to use App-schema Joining, the following configuration conditions must be met:

- All feature mappings used must be mapped to JDBC datastores.
- All feature mappings that are chained to each other must map to the same physical database.
- In your mappings, there are restrictions on the CQL expressions specified in the <SourceExpression> of both the referencing field in the parent feature as well as the referenced field in the nested feature (like FEATURE_LINK). Any operators or functions used in this expression must be supported by the filter capibilities, i.e. geotools must be able to translate them directly to SQL code. This can be different for each DBMS, though as a general rule it can assumed that comparison operators, logical operators and arithmetic operators are all supported but functions are not. Using simple field names for feature chaining is guaranteed to always work.

Failing to comply with any of these three restrictions when turning on Joining will result in exceptions thrown at run-time.

When using app-schema with Joining turned on, the following restrictions exist with respect to normal behaviour:

- XPaths specified inside Filters do not support handling referenced features (see *Multi-valued properties by reference (xlink:href)*) as if they were actual nested features, i.e. XPaths can only be evaluated when they can be evaluated against the actual XML code produced by WFS according to the XPath standard.

Configuration

Joining is turned on by default. It is disabled by adding this simple line to your app-schema.properties file (see *Property Interpolation*)

```
app-schema.joining = false
```

Or, alternatively, by setting the value of the Java System Property "app-schema.joining" to "false", for example

```
java -DGEOSERVER_DATA_DIR=... -Dapp-schema.joining=false Start
```

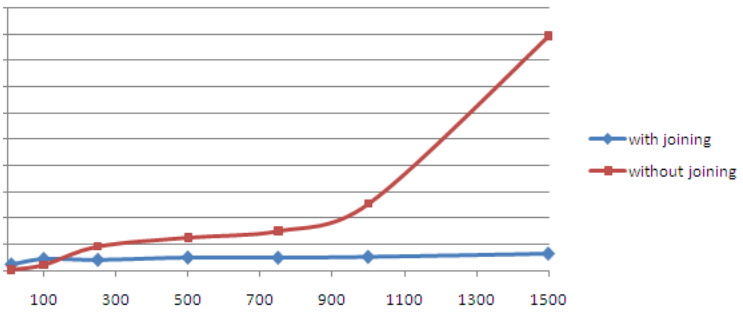
Not specifying "app-schema.joining" parameter will enable joining by default.

Database Design Guidelines

- Databases should be optimised for fast on-the-fly joining and ordering.
- Make sure to put indexes on all fields used as identifiers and for feature chaining, unique indexes where possible. Lack of indices may result in data being encoded in the wrong order or corrupted output when feature chaining is involved.
- Map your features preferably to normalised tables.
- It is recommended to apply feature chaining to regular one-to-many relationships, i.e. there should be a unique constraint defined on one of the fields used for the chaining, and if possible a foreign key constraint defined on the other field.

Effects on Performance

Typical curves of response time for configurations with and without joining against the amount of features produced will be shaped like this:



In the default implementation, response time increases rapidly with respect to the amount of produced features. This is because feature chaining is implemented by sending multiple SQL requests to the DBMS per feature, so the amount of requests increases with the amount of features produced. When Joining is turned on, response time will be almost constant with respect to the number of features. This is because

in this implementation a small amount of larger queries is sent to the DBMS, independant of the amount of features produced. In summary, difference in performance becomes greater as the amount of features requested gets bigger. General performance of joining will be dependant on database and mapping design (see above) and database size.

Using joining is strongly recommended when a large number of features need to be produced, for example when producing maps with WMS (see *WMS Support*).

Optimising the performance of the database will maximise the benefit of using joining, including for small queries.

Native Encoding of Filters on Nested Attributes

When App-Schema Joining is active, filters operating on nested attributes (i.e. attributes of features that are joined to the queried type via *Feature Chaining*) are translated to SQL and executed directly in the database backend, rather than being evaluated in memory after all features have been loaded (which was standard behavior in earlier versions of GeoServer). Native encoding can yield significant performance improvements, especially when the total number of features in the database is high (several thousands or more), but only a few of them would satisfy the filter.

There are, however, a few limitations in the current implementation:

1. Joining support must not have been explicitly disabled and all its pre-conditions must be met (see above)
2. Only binary comparison operators (e.g. `PropertyIsEqualTo`, `PropertyIsGreaterThan`, etc...), `PropertyIsLike` and `PropertyIsNull` filters are translated to SQL
3. Filters involving conditional polymorphic mappings are evaluated in memory
4. Filters comparing two or more different nested attributes are evaluated in memory
5. Filters matching multiple nested attribute mappings are evaluated in memory

Much like joining support, native encoding of nested filters is turned on by default, and it is disabled by adding to your app-schema.properties file the line

```
app-schema.encodeNestedFilters = false
```

Or, alternatively, by setting the value of the Java System Property "app-schema.encodeNestedFilters" to "false", for example

```
java -DGEOSERVER_DATA_DIR=... -Dapp-schema.encodeNestedFilters=false Start
```

UNION performance improvement for OR conditions

OR conditions are difficult to optimize for postgresql and are usually slow. App-Schema improves OR condition performance using UNION clauses instead OR for nested filter subqueries.

With UNION improvement enabled main OR binary operator on nested filter subquery will rebuild normal OR query like:

```
SELECT id, name FROM table WHERE name = "A" OR name = "B"
```

to:

```
SELECT id, name FROM table WHERE name = "A" UNION SELECT id, name FROM table WHERE␣
↪name = "B"
```

UNION improvement is enabled by default, and it is disabled by adding to your app-schema.properties file the line

```
app-schema.orUnionReplace = false
```

Or, alternatively, by setting the value of the Java System Property "app-schema.orUnionReplace" to "false", for example

```
java -DGEOSERVER_DATA_DIR=... -Dapp-schema.orUnionReplace=false Start
```

Note: This optimization will only be applied when a PostgresSQL database is being used.

5.6.18 Tutorial

This tutorial demonstrates how to configure two complex feature types using the app-schema plugin and data from two property files.

GeoSciML

This example uses Geoscience Markup Language (GeoSciML) 2.0, a GML 3.1 application schema:

> *"GeoSciML is an application schema that specifies a set of feature-types and supporting structures for information used in the solid-earth geosciences."*

The tutorial defines two feature types:

1. `gsml:GeologicUnit`, which describes "a body of material in the Earth".
2. `gsml:MappedFeature`, which describes the representation on a map of a feature, in this case `gsml:GeologicUnit`.

Because a single `gsml:GeologicUnit` can be observed at several distinct locations on the Earth's surface, it can have a multivalued `gsml:occurrence` property, each being a `gsml:MappedFeature`.

Installation

- Install GeoServer as usual.
- Install the app-schema plugin `geoserver-*-app-schema-plugin.zip`:
  - Place the jar files in `WEB-INF/lib`.
  - The `tutorial` folder contains the GeoServer configuraration (data directory) used for this tutorial.
    - Either replace your existing `data` directory with the tutorial data directory,
    - Or edit `WEB-INF/web.xml` to set `GEOSERVER_DATA_DIR` to point to the tutorial data directory. (Be sure to uncomment the section that sets `GEOSERVER_DATA_DIR`.)
- Perform any configuration required by your servlet container, and then start the servlet. For example, if you are using Tomcat, configure a new context in `server.xml` and then restart Tomcat.
- The first time GeoServer starts with the tutorial configuration, it will download all the schema (XSD) files it needs and store them in the `app-schema-cache` folder in the data directory. **You must be connected to the internet for this to work.**

datastore.xml

Each data store configuration file `datastore.xml` specifies the location of a mapping file and triggers its loading as an app-schema data source. This file should not be confused with the source data store, which is specified inside the mapping file.

For `gsml_GeologicUnit` the file is `workspaces/gsml/gsml_GeologicUnit/datastore.xml`:

```
<dataStore>
    <id>gsml_GeologicUnit_datastore</id>
    <name>gsml_GeologicUnit</name>
    <enabled>true</enabled>
    <workspace>
        <id>gsml_workspace</id>
    </workspace>
    <connectionParameters>
        <entry key="namespace">urn:cgi:xmlns:CGI:GeoSciML:2.0</entry>
        <entry key="url">file:workspaces/gsml/gsml_GeologicUnit/gsml_GeologicUnit.xml
↪</entry>
        <entry key="dbtype">app-schema</entry>
    </connectionParameters>
</dataStore>
```

For `gsml:MappedFeature` the file is `workspaces/gsml/gsml_MappedFeature/datastore.xml`:

```
<dataStore>
    <id>gsml_MappedFeature_datastore</id>
    <name>gsml_MappedFeature</name>
    <enabled>true</enabled>
    <workspace>
        <id>gsml_workspace</id>
    </workspace>
    <connectionParameters>
        <entry key="namespace">urn:cgi:xmlns:CGI:GeoSciML:2.0</entry>
        <entry key="url">file:workspaces/gsml/gsml_MappedFeature/gsml_MappedFeature.
↪xml</entry>
        <entry key="dbtype">app-schema</entry>
    </connectionParameters>
</dataStore>
```

Note: Ensure that there is no whitespace inside an `entry` element.

Mapping files

Configuration of app-schema feature types is performed in mapping files:

- `workspaces/gsml/gsml_GeologicUnit/gsml_GeologicUnit.xml`
- `workspaces/gsml/gsml_MappedFeature/gsml_MappedFeature.xml`

Namespaces

Each mapping file contains namespace prefix definitions:

```
<Namespace>
    <prefix>gml</prefix>
    <uri>http://www.opengis.net/gml</uri>
</Namespace>
<Namespace>
    <prefix>gsml</prefix>
    <uri>urn:cgi:xmlns:CGI:GeoSciML:2.0</uri>
</Namespace>
<Namespace>
    <prefix>xlink</prefix>
    <uri>http://www.w3.org/1999/xlink</uri>
</Namespace>
```

Only those namespace prefixes used in the mapping file need to be declared, so the mapping file for `gsml:GeologicUnit` has less.

Source data store

The data for this tutorial is contained in two property files:

- `workspaces/gsml/gsml_GeologicUnit/gsml_GeologicUnit.properties`
- `workspaces/gsml/gsml_MappedFeature/gsml_MappedFeature.properties`

Java Properties describes the format of property files.

For this example, each feature type uses an identical source data store configuration. This `directory` parameter indicates that the source data is contained in property files named by their feature type, in the same directory as the corresponding mapping file:

```
<sourceDataStores>
     <DataStore>
         <id>datastore</id>
         <parameters>
             <Parameter>
                 <name>directory</name>
                 <value>file:./</value>
             </Parameter>
         </parameters>
     </DataStore>
 </sourceDataStores>
```

See *Data Stores* for a description of how to use other types of data stores such as databases.

Target types

Both feature types are defined by the same XML Schema, the top-level schema for GeoSciML 2.0. This is specified in the `targetTypes` section. The type of the output feature is defined in `targetElement` in the `typeMapping` section below:

```
<targetTypes>
    <FeatureType>
        <schemaUri>http://www.geosciml.org/geosciml/2.0/xsd/geosciml.xsd</schemaUri>
    </FeatureType>
</targetTypes>
```

In this case the schema is published, but because the OASIS XML Catalog is used for schema resolution, a private or modified schema in the catalog can be used if desired.

Mappings

The `typeMappings` element begins with configuration elements. From the mapping file for `gsml:GeologicUnit`:

```
<typeMappings>
    <FeatureTypeMapping>
        <sourceDataStore>datastore</sourceDataStore>
        <sourceType>gsml_GeologicUnit</sourceType>
        <targetElement>gsml:GeologicUnit</targetElement>
```

- The mapping starts with `sourceDataStore`, which gives the arbitrary identifier used above to name the source of the input data in the `sourceDataStores` section.
- `sourceType` gives the name of the source simple feature type. In this case it is the simple feature type `gsml_GeologicUnit`, sourced from the rows of the file `gsml_GeologicUnit.properties` in the same directory as the mapping file.
- When working with databases `sourceType` is the name of a table or view. Database identifiers must be lowercase for PostGIS or uppercase for Oracle Spatial.
- `targetElement` is the name of the output complex feature type.

gml:id mapping

The first mapping sets the `gml:id` to be the feature id specified in the source property file:

```
<AttributeMapping>
    <targetAttribute>
        gsml:GeologicUnit
    </targetAttribute>
    <idExpression>
        <OCQL>ID</OCQL>
    </idExpression>
</AttributeMapping>
```

- `targetAttribute` is the XPath to the element for which the mapping applies, in this case, the top-level feature type.
- `idExpression` is a special form that can only be used to set the `gml:id` on a feature. Any field or CQL expression can be used, if it evaluates to an NCName.

Ordinary mapping

Most mappings consist of a target and source. Here is one from `gsml:GeologicUnit`:

```
<AttributeMapping>
    <targetAttribute>
        gml:description
        </targetAttribute>
    <sourceExpression>
        <OCQL>DESCRIPTION</OCQL>
```

```
    </sourceExpression>
</AttributeMapping>
```

- In this case, the value of `gml:description` is just the value of the `DESCRIPTION` field in the property file.
- For a database, the field name is the name of the column (the table/view is set in `sourceType` above). Database identifiers must be lowercase for PostGIS or uppercase for Oracle Spatial.
- CQL expressions can be used to calculate content. Use caution because queries on CQL-calculated values prevent the construction of efficient SQL queries.
- Source expressions can be CQL literals, which are single-quoted.

Client properties

In addition to the element content, a mapping can set one or more "client properties" (XML attributes). Here is one from `gsml:MappedFeature`:

```
<AttributeMapping>
    <targetAttribute>
        gsml:specification
    </targetAttribute>
    <ClientProperty>
        <name>xlink:href</name>
        <value>GU_URN</value>
    </ClientProperty>
</AttributeMapping>
```

- This mapping leaves the content of the `gsml:specification` element empty but sets an `xlink:href` attribute to the value of the `GU_URN` field.
- Multiple `ClientProperty` mappings can be set.

In this example from the mapping for `gsml:GeologicUnit` both element content and an XML attribute are provided:

```
<AttributeMapping>
    <targetAttribute>
        gml:name[1]
        </targetAttribute>
    <sourceExpression>
        <OCQL>NAME</OCQL>
    </sourceExpression>
    <ClientProperty>
        <name>codeSpace</name>
        <value>'urn:x-test:classifierScheme:TestAuthority:GeologicUnitName'</value>
    </ClientProperty>
</AttributeMapping>
```

- The `codespace` XML attribute is set to a fixed value by providing a CQL literal.
- There are multiple mappings for `gml:name`, and the index `[1]` means that this mapping targets the first.

targetAttributeNode

If the type of a property is abstract, a `targetAttributeNode` mapping must be used to specify a concrete type. This mapping must occur before the mapping for the content of the property.

Here is an example from the mapping file for `gsml:MappedFeature`:

```
<AttributeMapping>
    <targetAttribute>gsml:positionalAccuracy</targetAttribute>
    <targetAttributeNode>gsml:CGI_TermValuePropertyType</targetAttributeNode>
</AttributeMapping>
<AttributeMapping>
    <targetAttribute>gsml:positionalAccuracy/gsml:CGI_TermValue/gsml:value</
↪targetAttribute>
    <sourceExpression>
        <OCQL>'urn:ogc:def:nil:OGC:missing'</OCQL>
    </sourceExpression>
    <ClientProperty>
        <name>codeSpace</name>
        <value>'urn:ietf:rfc:2141'</value>
    </ClientProperty>
</AttributeMapping>
```

- `gsml:positionalAccuracy` is of type `gsml:CGI_TermValuePropertyType`, which is abstract, so must be mapped to its concrete subtype `gsml:CGI_TermValuePropertyType` with a `targetAttributeNode` mapping before its contents can be mapped.
- This example also demonstrates that mapping can be applied to nested properties to arbitrary depth. This becomes unmanageable for deep nesting, where feature chaining is preferred.

Feature chaining

In feature chaining, one feature type is used as a property of an enclosing feature type, by value or by reference:

```
<AttributeMapping>
    <targetAttribute>
        gsml:occurrence
    </targetAttribute>
    <sourceExpression>
        <OCQL>URN</OCQL>
        <linkElement>gsml:MappedFeature</linkElement>
        <linkField>gml:name[2]</linkField>
    </sourceExpression>
    <isMultiple>true</isMultiple>
</AttributeMapping>
```

- In this case from the mapping for `gsml:GeologicUnit`, we specify a mapping for its `gsml:occurrence`.
- The `URN` field of the source `gsml_GeologicUnit` simple feature is use as the "foreign key", which maps to the second `gml:name` in each `gsml:MappedFeature`.
- Every `gsml:MappedFeature` with `gml:name[2]` equal to the `URN` of the `gsml:GeologicUnit` under construction is included as a `gsml:occurrence` property of the `gsml:GeologicUnit` (by value).

WFS response

When GeoServer is running, test app-schema WFS in a web browser. If GeoServer is listening on `localhost:8080` you can query the two feature types using these links:

- http://localhost:8080/geoserver/wfs?request=GetFeature&version=1.1.0&typeName=gsml:GeologicUnit
- http://localhost:8080/geoserver/wfs?request=GetFeature&version=1.1.0&typeName=gsml:MappedFeature

gsml:GeologicUnit

Feature chaining has been used to construct the multivalued property `gsml:occurrence` of `gsml:GeologicUnit`. This property is a `gsml:MappedFeature`. The WFS response for `gsml:GeologicUnit` combines the output of both feature types into a single response. The first `gsml:GeologicUnit` has two `gsml:occurrence` properties, while the second has one. The relationships between the feature instances are data driven.

Because the mapping files in the tutorial configuration do not contain attribute mappings for all mandatory properties of these feature types, the WFS response is not *schema-valid* against the GeoSciML 2.0 schemas. Schema-validity can be achieved by adding more attribute mappings to the mapping files.

Note: These feature types are defined in terms of GML 3.1 (the default for WFS 1.1.0); other GML versions will not work.

Warning: The web interface does not yet support app-schema store or layer administration.

Acknowledgements

`gsml_GeologicUnit.properties` and `gsml_MappedFeature.properties` are derived from data provided by the Department of Primary Industries, Victoria, Australia. For the purposes of this tutorial, this data has been modified to the extent that it has no real-world meaning.

5.6.19 MongoDB Tutorial

This tutorial demonstrates how to use app-schema plugin with a MongoDB data store. This tutorial will focus on the MongoDB data store specificities is highly recommended to read the app-schema documentation before.

Use Case

The use case for this tutorial will be to serve through app-schema the information about some meteorological stations stored in a MongoDB database. Note that this use case is completely fictional and only used to demonstrate the MongoDB and app-schema integration.

First of all let's insert some test data in a MongoDB data store:

```
db.stations.insert({
    "id": "1",
    "name": "station 1",
    "contact": {
        "mail": "station1@mail.com"
    },
    "geometry": {
        "coordinates": [
            50,
            60
        ],
        "type": "Point"
    },
    "measurements": [
        {
            "name": "temp",
            "unit": "c",
            "values": [
                {
                    "time": 1482146800,
                    "value": 20
                }
            ]
        },
        {
            "name": "wind",
            "unit": "km/h",
            "values": [
                {
                    "time": 1482146833,
                    "value": 155
                }
            ]
        }
    ]
})

db.stations.insert({
    "id": "2",
    "name": "station 2",
    "contact": {
        "mail": "station2@mail.com"
    },
    "geometry": {
        "coordinates": [
            100,
            -50
        ],
        "type": "Point"
    },
    "measurements": [
        {
            "name": "temp",
            "unit": "c",
            "values": [
                {
                    "time": 1482146911,
                    "value": 35
```

```
                },
                {
                    "time": 1482146935,
                    "value": 25
                }
            ]
        },
        {
            "name": "wind",
            "unit": "km/h",
            "values": [
                {
                    "time": 1482146964,
                    "value": 80
                }
            ]
        },
        {
            "name": "pression",
            "unit": "pa",
            "values": [
                {
                    "time": 1482147026,
                    "value": 1019
                },
                {
                    "time": 1482147051,
                    "value": 1015
                }
            ]
        }
    ]
})

db.stations.createIndex({
    "geometry": "2dsphere"
})
```

This is the schema that will be used to do the mappings in app-schema:

```
<xs:schema version="1.0" xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:gml="http://www.opengis.net/gml"
           xmlns:st="http://www.stations.org/1.0"
           targetNamespace="http://www.stations.org/1.0"
           elementFormDefault="qualified" attributeFormDefault="unqualified">

  <xs:import namespace="http://www.opengis.net/gml"
             schemaLocation="http://schemas.opengis.net/gml/3.2.1/gml.xsd"/>

  <xs:complexType name="ContactType">
    <xs:sequence>
      <xs:element name="mail" minOccurs="0" maxOccurs="1" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="MeasurementPropertyType">
    <xs:sequence minOccurs="0">
      <xs:element ref="st:Measurement"/>
```

```
    </xs:sequence>
    <xs:attributeGroup ref="gml:AssociationAttributeGroup"/>
  </xs:complexType>

  <xs:complexType name="MeasurementType" abstract="true">
    <xs:sequence>
      <xs:element name="name" minOccurs="1" maxOccurs="1" type="xs:string"/>
      <xs:element name="unit" minOccurs="1" maxOccurs="1" type="xs:string"/>
      <xs:element name="values" minOccurs="1" maxOccurs="unbounded" type=
↪"st:ValuePropertyType"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="ValuePropertyType">
    <xs:sequence minOccurs="0">
      <xs:element ref="st:Value"/>
    </xs:sequence>
    <xs:attributeGroup ref="gml:AssociationAttributeGroup"/>
  </xs:complexType>

  <xs:complexType name="ValueType">
    <xs:sequence>
      <xs:element name="timestamp" minOccurs="1" maxOccurs="1" type="xs:long"/>
      <xs:element name="value" minOccurs="1" maxOccurs="1" type="xs:double"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="StationFeatureType">
    <xs:complexContent>
      <xs:extension base="gml:AbstractFeatureType">
        <xs:sequence>
          <xs:element name="name" minOccurs="1" maxOccurs="1" type="xs:string"/>
          <xs:element name="contact" minOccurs="0" maxOccurs="1" type="st:ContactType
↪"/>
          <xs:element name="measurement" minOccurs="0" maxOccurs="unbounded" type=
↪"st:MeasurementPropertyType"/>
          <xs:element name="geometry" type="gml:GeometryPropertyType" minOccurs="0"␣
↪maxOccurs="1"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <xs:element name="StationFeature" type="st:StationFeatureType"  substitutionGroup=
↪"gml:_Feature"/>
  <xs:element name="Measurement" type="st:MeasurementType"  substitutionGroup="gml:_
↪Feature"/>
  <xs:element name="Value" type="st:ValueType"  substitutionGroup="gml:_Feature"/>

</xs:schema>
```

Mappings

MongoDB objects may contain nested elements and nested collections. The following three functions make possible to select nested elements and link nested collections using a JSON path:

Function	Example	Description
jsonSelect	jsonSelect('contact.mail')	Used to retrieve the value for the mapping from a MongoDB object.
collectionLink	collectionLink('measurements.values')	Used when chaining entities with a nested collection.
collectionId	collectionId()	Instructs the mapper to generate a ID for the nested collection.
nestedCollectionLink	nestedCollectionLink()	Used on the nested collection to create a link with the parent feature.

A station data is composed of some meta-information about the station and a list of measurements. Each measurement as some meta-information and contains a list of values. The mappings will contain three top entities: the station, the measurements and the values.

Follows a the complete mappings file:

```
<?xml version="1.0" encoding="UTF-8"?>
<as:AppSchemaDataAccess xmlns:as="http://www.geotools.org/app-schema"
                        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                        xsi:schemaLocation="http://www.geotools.org/app-schema␣
↪AppSchemaDataAccess.xsd">
<namespaces>
  <Namespace>
    <prefix>st</prefix>
    <uri>http://www.stations.org/1.0</uri>
  </Namespace>
  <Namespace>
    <prefix>gml</prefix>
    <uri>http://www.opengis.net/gml</uri>
  </Namespace>
</namespaces>

<sourceDataStores>
  <DataStore>
    <id>data_source</id>
    <parameters>
      <Parameter>
        <name>data_store</name>
        <value>mongodb://{mongoHost}:{mongoPort}/{dataBaseName}</value>
      </Parameter>
      <Parameter>
        <name>namespace</name>
        <value>http://www.stations.org/1.0</value>
      </Parameter>
      <Parameter>
        <name>schema_store</name>
        <value>file:{schemaStore}</value>
      </Parameter>
      <Parameter>
        <name>data_store_type</name>
        <value>complex</value>
      </Parameter>
    </parameters>
  </DataStore>
</sourceDataStores>

<targetTypes>
```

```
  <FeatureType>
    <schemaUri>stations.xsd</schemaUri>
  </FeatureType>
</targetTypes>

<typeMappings>
  <FeatureTypeMapping>
    <sourceDataStore>data_source</sourceDataStore>
    <sourceType>{collectionName}</sourceType>
    <targetElement>st:StationFeature</targetElement>
    <attributeMappings>
      <AttributeMapping>
        <targetAttribute>st:StationFeature</targetAttribute>
        <idExpression>
          <OCQL>jsonSelect('id')</OCQL>
        </idExpression>
      </AttributeMapping>
      <AttributeMapping>
        <targetAttribute>st:name</targetAttribute>
        <sourceExpression>
          <OCQL>jsonSelect('name')</OCQL>
        </sourceExpression>
      </AttributeMapping>
      <AttributeMapping>
        <targetAttribute>st:contact/st:mail</targetAttribute>
        <sourceExpression>
          <OCQL>jsonSelect('contact.mail')</OCQL>
        </sourceExpression>
      </AttributeMapping>
      <AttributeMapping>
        <targetAttribute>st:measurement</targetAttribute>
        <sourceExpression>
          <OCQL>collectionLink('measurements')</OCQL>
          <linkElement>aaa</linkElement>
          <linkField>FEATURE_LINK[1]</linkField>
        </sourceExpression>
        <isMultiple>true</isMultiple>
      </AttributeMapping>
      <AttributeMapping>
        <targetAttribute>st:geometry</targetAttribute>
        <sourceExpression>
          <OCQL>jsonSelect('geometry')</OCQL>
        </sourceExpression>
      </AttributeMapping>
    </attributeMappings>
  </FeatureTypeMapping>
  <FeatureTypeMapping>
    <sourceDataStore>data_source</sourceDataStore>
    <sourceType>{collectionName}</sourceType>
    <mappingName>aaa</mappingName>
    <targetElement>st:Measurement</targetElement>
    <attributeMappings>
      <AttributeMapping>
        <targetAttribute>st:Measurement</targetAttribute>
        <idExpression>
          <OCQL>collectionId()</OCQL>
        </idExpression>
      </AttributeMapping>
```

```
      <AttributeMapping>
        <targetAttribute>st:name</targetAttribute>
        <sourceExpression>
          <OCQL>jsonSelect('name')</OCQL>
        </sourceExpression>
      </AttributeMapping>
      <AttributeMapping>
        <targetAttribute>st:unit</targetAttribute>
        <sourceExpression>
          <OCQL>jsonSelect('unit')</OCQL>
        </sourceExpression>
      </AttributeMapping>
      <AttributeMapping>
        <targetAttribute>st:values</targetAttribute>
        <sourceExpression>
          <OCQL>collectionLink('values')</OCQL>
          <linkElement>st:Value</linkElement>
          <linkField>FEATURE_LINK[2]</linkField>
        </sourceExpression>
        <isMultiple>true</isMultiple>
      </AttributeMapping>
      <AttributeMapping>
        <targetAttribute>FEATURE_LINK[1]</targetAttribute>
        <sourceExpression>
          <OCQL>nestedCollectionLink()</OCQL>
        </sourceExpression>
      </AttributeMapping>
    </attributeMappings>
  </FeatureTypeMapping>
  <FeatureTypeMapping>
    <sourceDataStore>data_source</sourceDataStore>
    <sourceType>{collectionName}</sourceType>
    <targetElement>st:Value</targetElement>
    <attributeMappings>
      <AttributeMapping>
        <targetAttribute>st:Value</targetAttribute>
        <idExpression>
          <OCQL>collectionId()</OCQL>
        </idExpression>
      </AttributeMapping>
      <AttributeMapping>
        <targetAttribute>st:timestamp</targetAttribute>
        <sourceExpression>
          <OCQL>jsonSelect('time')</OCQL>
        </sourceExpression>
      </AttributeMapping>
      <AttributeMapping>
        <targetAttribute>st:value</targetAttribute>
        <sourceExpression>
          <OCQL>jsonSelect('value')</OCQL>
        </sourceExpression>
      </AttributeMapping>
      <AttributeMapping>
        <targetAttribute>FEATURE_LINK[2]</targetAttribute>
        <sourceExpression>
          <OCQL>nestedCollectionLink()</OCQL>
        </sourceExpression>
      </AttributeMapping>
```

```
    </attributeMappings>
  </FeatureTypeMapping>
</typeMappings>

</as:AppSchemaDataAccess>
```

The mappings for the attributes are straightforward, for example the following mapping:

```
<AttributeMapping>
    <targetAttribute>st:contact/st:mail</targetAttribute>
    <sourceExpression>
        <OCQL>jsonSelect('contact.mail')</OCQL>
    </sourceExpression>
</AttributeMapping>
```

The mapping above defines that the contact mail for a station will be available at the JSON path `contact.mail` and that the correspondent XML schema element is the XPATH `st:contact/st:mail`.

The feature chaining is a little bit more complex. Let's take as an example the chaining between `StationFeature` and `Measurement` features. In the `StationFeature` feature type the link to the Measurement entity is defined with the following mapping:

```
<AttributeMapping>
    <targetAttribute>st:measurement</targetAttribute>
    <sourceExpression>
        <OCQL>collectionLink('measurements')</OCQL>
        <linkElement>st:Measurement</linkElement>
        <linkField>FEATURE_LINK[1]</linkField>
    </sourceExpression>
    <isMultiple>true</isMultiple>
</AttributeMapping>
```

and in the `Measurement` feature type the link to the parent feature is defined with the following mapping:

```
<AttributeMapping>
    <targetAttribute>FEATURE_LINK[1]</targetAttribute>
    <sourceExpression>
        <OCQL>nestedCollectionLink()</OCQL>
    </sourceExpression>
</AttributeMapping>
```

With the two mapping above we tie the two features types together. When working with a MongoDB data store this mappings will always be petty much the same, only the nested collection path and the feature link index need to be updated. Note that the JSON path of the nested collections attributes are relative to the parent.

Querying

To create an MongoDB app-schema layer in GeoServer, the app-schema extension and the mongo-complex extension needs to be installed.

A workspace for each name space declared in the mappings file needs to be created, in this case the workspace `st` with URI `http://www.stations.org/1.0` needs to be created. No need to create a `gml` workspace.

Creating a MongoDB app-schema layer is similar to any other app-schema layer, just create an app-schema store pointing to the correct mappings file and select the layer correspondent to the top entity, in this case

`st:StationFeature`.

Is possible to query with WFS complex features encoded in GML and GeoJson using complex features filtering capabilities. For example, querying all the stations that have a measurement value with a time stamp superior to `1482146964`:

```
<wfs:Query typeName="st:StationFeature">
    <ogc:Filter>
        <ogc:Filter>
            <ogc:PropertyIsGreaterThan>
                    <ogc:PropertyName>
                        st:StationFeature/st:measurement/st:values/st:timestamp
                    </ogc:PropertyName>
                    <ogc:Literal>
                        1482146964
                    </ogc:Literal>
                </ogc:PropertyIsGreaterThan>
        </ogc:Filter>
    </ogc:Filter>
</wfs:Query>
```

5.6.20 Apache Solr Tutorial

This tutorial demonstrates how to use the App-Schema plugin with a Apache Solr data store. This tutorial will focus on the Apache Solr data store specific aspects, and the App-Schema documentation should be read first.

The use case for this tutorial will be to serve through App-Schema the information about some meteorological stations index in an Apache Solr core. Note that this use case is completely fictional and only used to demonstrate the Apache Solr and App-Schema integration.

A station data is composed of some meta-information about the station, e.g. it's name and position. The only extra different configuration we need to provide when using Apache Solr as a data source is the configuration of the data store itself.

Apache Solr data source configuration as a specific syntax and allow us to specify geometry attributes and to explicitly set the default geometry:

```
<sourceDataStores>
  <SolrDataStore>
    <id>stations</id>
    <url>http://localhost:8983/solr/stations</url>
    <index name="stations">
      <geometry default="true">
        <name>location</name>
        <srid>4326</srid>
        <type>POINT</type>
      </geometry>
    </index>
  </SolrDataStore>
</sourceDataStores>
```

In this particular case the the `location` attribute contains a point geometry and will be the default geometry.

The complete mapping file is:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<as:AppSchemaDataAccess xmlns:as="http://www.geotools.org/app-schema">
    <namespaces>
        <Namespace>
            <prefix>st</prefix>
            <uri>http://www.stations.org/1.0</uri>
        </Namespace>
        <Namespace>
            <prefix>gml</prefix>
            <uri>http://www.opengis.net/gml/3.2</uri>
        </Namespace>
    </namespaces>
    <sourceDataStores>
        <SolrDataStore>
            <id>stations</id>
            <url>http://localhost:8983/solr/stations</url>
            <index name="stations">
                <geometry default="true">
                    <name>location</name>
                    <srid>4326</srid>
                    <type>POINT</type>
                </geometry>
            </index>
        </SolrDataStore>
    </sourceDataStores>
    <targetTypes>
        <FeatureType>
            <schemaUri>stations.xsd</schemaUri>
        </FeatureType>
    </targetTypes>
    <typeMappings>
        <FeatureTypeMapping>
            <mappingName>stations_solr</mappingName>
            <sourceDataStore>stations</sourceDataStore>
            <sourceType>stations</sourceType>
            <targetElement>st:Station</targetElement>
            <attributeMappings>
                <AttributeMapping>
                    <targetAttribute>st:Station</targetAttribute>
                    <idExpression>
                        <OCQL>station_id</OCQL>
                    </idExpression>
                </AttributeMapping>
                <AttributeMapping>
                    <targetAttribute>st:stationName</targetAttribute>
                    <sourceExpression>
                        <OCQL>station_name</OCQL>
                    </sourceExpression>
                </AttributeMapping>
                <AttributeMapping>
                    <targetAttribute>st:position</targetAttribute>
                    <sourceExpression>
                        <OCQL>station_location</OCQL>
                    </sourceExpression>
                </AttributeMapping>
            </attributeMappings>
        </FeatureTypeMapping>
    </typeMappings>
```

```
</as:AppSchemaDataAccess>
```

The mappings for the attributes are straightforward and follow the normal App-Schema attributes mappings syntax. Currently multi valued fields are not supported.

CHAPTER 6

Styling

This section discusses the styling of geospatial data served through GeoServer.

6.1 Styles

This section will detail how to work with the styles pages in the *Web administration interface*. For more information on styles and syntax, please see the main section on *Styling*.

Styles are used to control the appearance of geospatial data. Styles for GeoServer are written in a number of different formats:

- **Styled Layer Descriptor (SLD)**: An OGC standard for geospatial styling. Available by default.
- **Cascading Style Sheets (CSS)**: A CSS-like syntax. Available via an *extension*.
- **YSLD**: An SLD-equivalent based on YAML for improved authoring. Available via the *ysld extension* .
- **MBStyle**: A syntax based on JSON for improved interoperability. Available via the *mbstyle extension* .

6.1.1 Styles page

On the Styles page, you can *add a new style*, *remove a style*, or *view or edit an existing style*.

Add a Style

The buttons for adding and removing a style can be found at the top of the *Styles* page.

To add a new style, click *Add a new style* button. You will be redirected to the new style page, which is the same as the Style Editor *Data* tab.

The editor page provides several options for submitting a new style:

- **Type** the style definition directly into the editor.

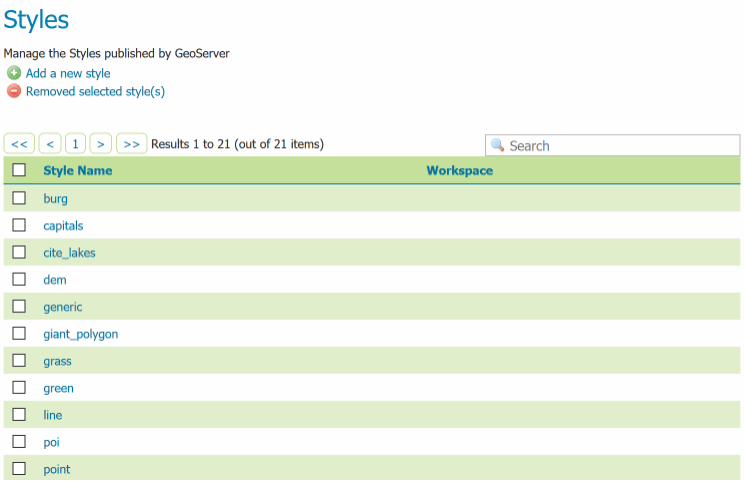


Fig. 6.1: Styles page

Fig. 6.2: Adding or removing a style

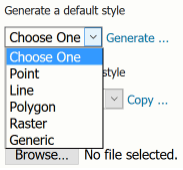


Fig. 6.3: Generating a new default style.

- **Generate** a new default style based on an internal template:
- **Copy** the contents of an existing style into the editor:

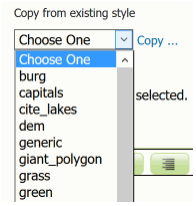


Fig. 6.4: Copying an existing Style from GeoServer

- **Upload** a local file that contains the style:

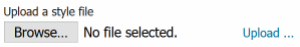


Fig. 6.5: Uploading an file from the local system

When creating a style, only the *Data* tab will be available. Click *Apply* on the new style to stay on the Style Editor page and gain access to all tabs.

Remove a Style

To remove a style, click the check box next to the style. Multiple styles can be selected at the same time. Click the *Remove selected style(s)* link at the top of the page. You will be asked for confirmation:

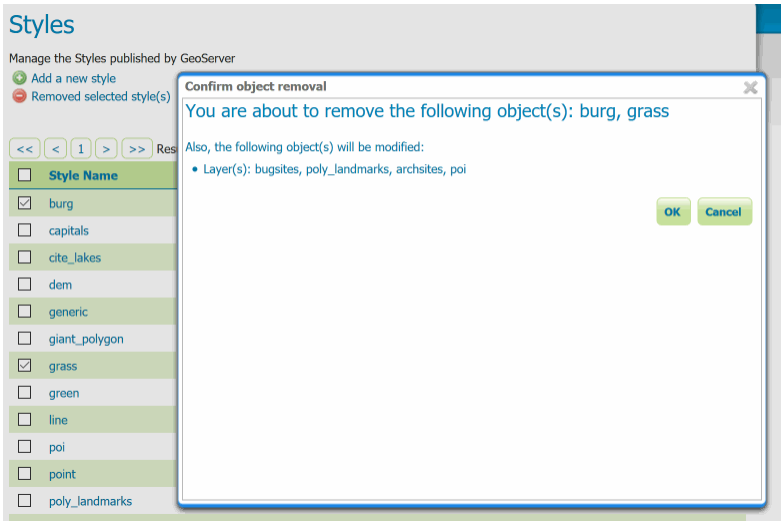


Fig. 6.6: Confirmation prompt for removing styles

Click *OK* to remove the selected style(s).

6.1.2 Style Editor

On the Styles page, click a style name to open the *Style Editor*.

The Style Editor page presents the *style definition*. The page contains four tabs with many configuration options:

- *Data*: Includes basic style information, the ability to generate a style, and legend details
- *Publishing*: Displays which layers are using this style
- *Layer Preview*: Previews the style with an associated layer while editing
- *Layer Attributes*: Displays a list of attributes for the associated layer

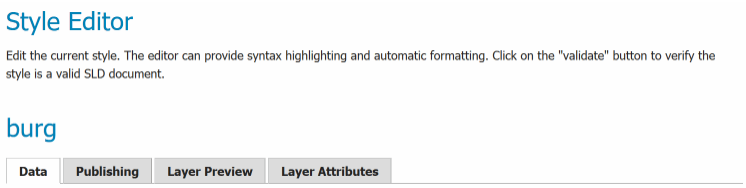


Fig. 6.7: Style Editor tabs

At the bottom of the Style Editor page is a number of options:

Option	Description
Validate	Will test the current style for correctness according to the *Format* option selected
Apply	Makes the changes to the style and remain on the Style Editor page. This is useful to update the *Layer Preview* tab.
Submit	Makes the changes to the style and returns to the Styles page
Cancel	Cancels all changes to the style and returns to the Styles page

Fig. 6.8: Style Editor options

Style definition

On all tabs, the Style Editor will display the style definition at the bottom, allowing for direct editing of the style. Switch between the tabs in order to facilitate style creation and editing.

The style editor supports line numbering, automatic indentation, and real-time syntax highlighting. You can also increase or decrease the font size of the editor.

Button	Description
	Undo
	Redo
	Go to line
	Auto-format the editor contents
	Change the font size in the editor
	Insert image into style (choose existing or upload)
	Change height of style editor (disabled in full screen mode)

During editing and especially after editing is complete, you will want to check validation of the syntax. This can be done by clicking the *Validate* button at the bottom.

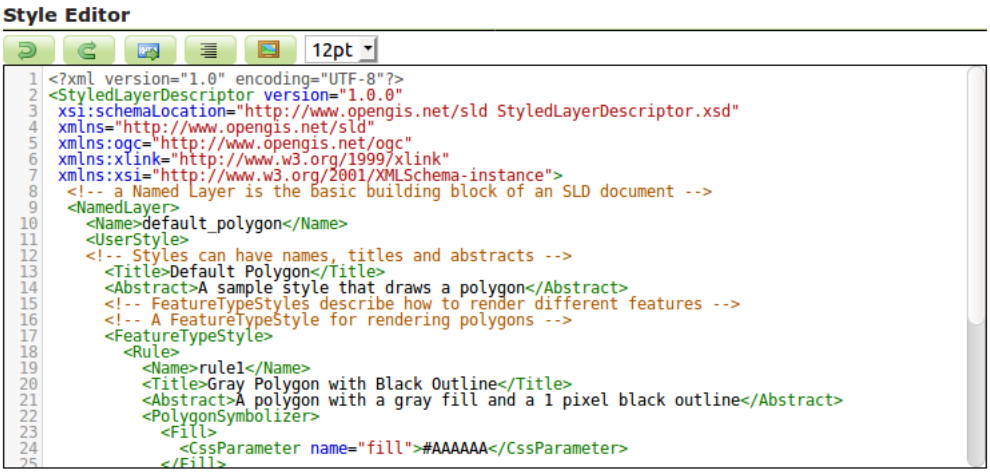


Fig. 6.9: Style editor

If no errors are found, you will see this message:

Fig. 6.10: No validation errors

If any validation errors are found, they will be displayed:

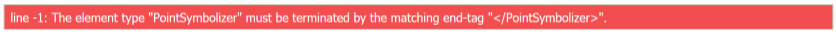


Fig. 6.11: Validation error message

Style Editor: Data tab

The Data tab includes basic style information, the ability to generate a style, and legend details.

The *Style Data* area has mandatory basic style information:

Option	Description
Name	Name of the style
Workspace	Workspace in which the style is contained. Styles can be inside workspaces, but can also be "global" (no workspace).
Format	Format of the style. Options are *SLD*, *CSS*, and *YSLD*, *MBStyle* depending on availability.

The *Style Content* area allows you to generate a style, copy an existing style, or upload an existing style:

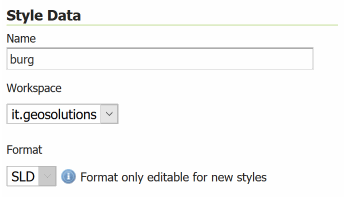


Fig. 6.12: Style Data area

Option	Description
Generate a default style	Selects a generic style based on geometry. Options are *Point*, *Line*, *Polygon*, *Raster*, and *Generic*. Click *Generate* when selected.
Copy from existing style	Selects an existing style in GeoServer and copy its contents to this style. Any style in GeoServer is available as an option. Not all styles will work with all layers. Click *Copy* when selected.
Upload a style file	Selects a plain text file on your local system to add as the style. Click *Upload* when selected.

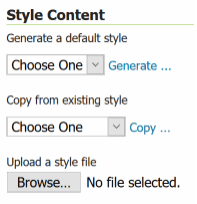


Fig. 6.13: Style Content area

The *Legend* area allows you to add, modify, or delete a custom style, and preview the legend for the style. By default GeoServer will generate a legend based on your style file, but this can be customized here:

Option	Description
Add legend	Allows you to use a custom legend
Online Resource	Path to the custom legend graphic to use. Can be a URL or a local path (relative to the style file path). See *Structure of the data directory* for a description of the styles directory.
Auto-detect image size and type	Populates the *Width*, *Height*, and *Format* options for the *Online Resource*
Width	Width of the custom legend graphic
Height	Height of the custom legend graphic
Format	Mime type of the custom legend graphic
Discard legend	Will remove the settings for the custom legend graphic and will instead use the default generated legend.
Preview legend	Previews the legend based on the current settings

Style Editor: Publishing tab

The Publishing tab displays a list of all layers on the server, with the purpose of showing which layers are associated with the current style. Layers can set a single default style and have any number of additional

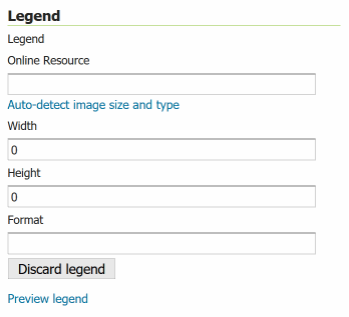


Fig. 6.14: Legend area

styles. If this style is set to be either of these options for a layer, it will be shown with a check box in the table.

Option	Description
Workspace	Workspace of the layer
Layer	Name of the layer
Default	Shows whether the style being edited is the default for a given layer
Associated	Shows whether the style being edited is an additional style for a given layer

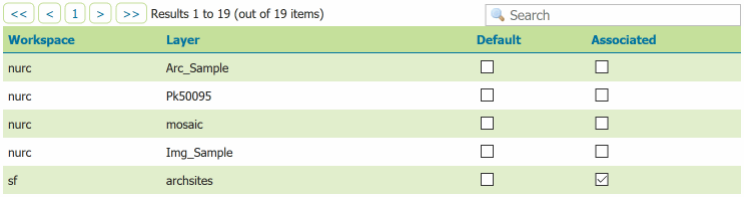


Fig. 6.15: Publishing tab

Style Editor: Layer Preview tab

It is very common to have to iterate your styles and test how the visualization changes over time. The Layer Preview tab allows you to make changes to the style and see them without having to navigate away from the page.

The Layer Preview tab shows a single image. GeoServer tries to identify which layer should be shown (for example, a layer for which this style is the default), but if the layer being previewed is not the desired one, click the layer name above the preview box and select a layer.

Style Editor: Layer Attributes tab

Most styles utilize the specific values of certain attributes of the associated layer in order to create more detailed and useful styles. (For example: styling all large cities different from small cities based on a particular attribute.)

The Layer Attributes tab will display a list of attributes for the given associated layer. GeoServer tries to identify which layer should be shown (for example, a layer for which this style is the default), but if the layer being previewed is not the desired one, click the layer name above the table and select a layer.

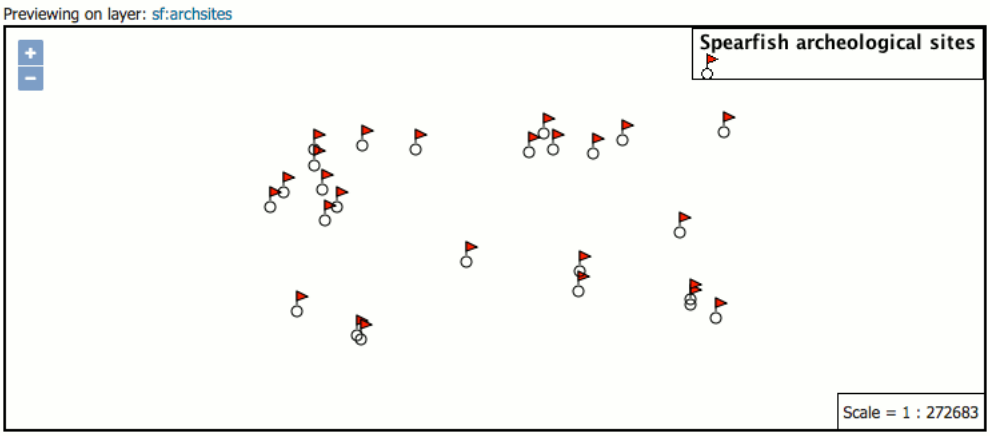


Fig. 6.16: Layer Preview tab

Option	Description
name	Name of the attribute
type	Type of the attribute. Can be a numeric (such as "Long"), a string ("String"), or a geometry (such as "Point").
sample	Sample value of the attribute taken from the data
min	Minimum value of the attribute in the data set, if applicable
max	Minimum value of the attribute in the data set, if applicable
computeStats	Click *Compute* to calculate the *min* and *max* values for that attribute, if applicable

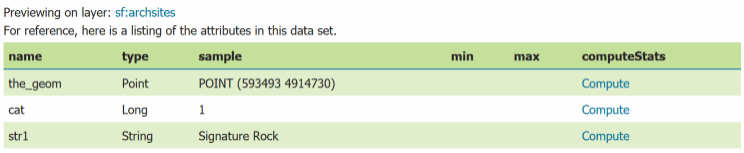


Fig. 6.17: Layer Attributes tab

Style Editor: full screen side by side mode

The style editor page has now a "outwards arrows" button on the top right side of the window:

Pressing it will cause the editor and preview to go side by side and use the entire browser window space:

The button turns into a "inwards arrows" icon, pressing it resumes the original editing mode.

6.2 SLD Styling

This section discusses styling of geospatial data using "Style Layer Descriptor" XML files.

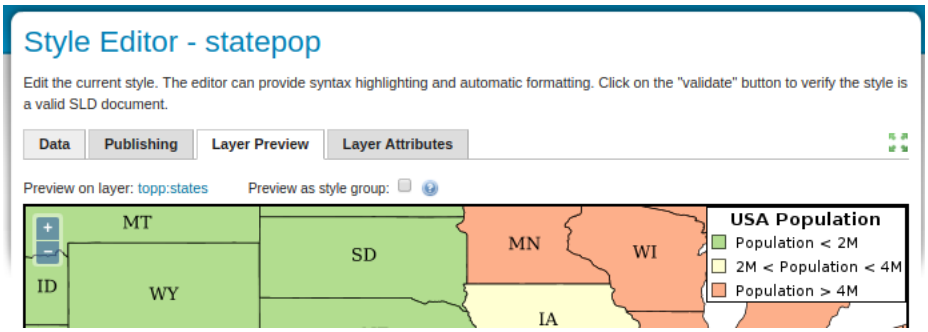


Fig. 6.18: The new fullscreen functionality

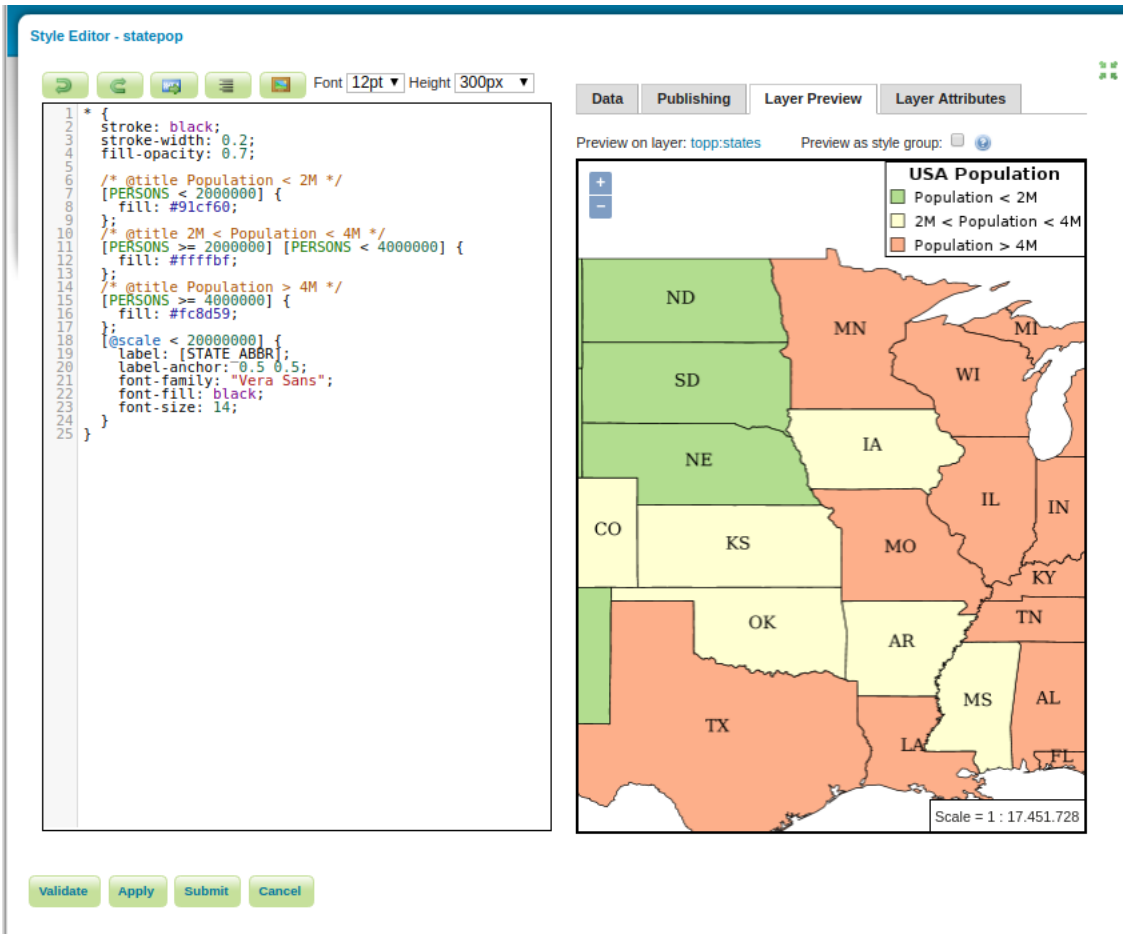


Fig. 6.19: Side by side style editing

6.2.1 Introduction to SLD

Geospatial data has no intrinsic visual component. In order to see data, it must be styled. Styling specifies color, thickness, and other visible attributes used to render data on a map.

In GeoServer, styling is accomplished using a markup language called Styled Layer Descriptor, or SLD for short. SLD is an XML-based markup language and is very powerful, although somewhat complex. This page gives an introduction to the capabilities of SLD and how it works within GeoServer.

Note: Since GeoServer uses SLD exclusively for styling, the terms "SLD" and "style" will often be used interchangeably.

SLD Concepts

In GeoServer styling is most often specified using XML **SLD style documents**. Style documents are associated with GeoServer **layers** (**featuretypes**) to specify how they should be **rendered**. A style document specifies a single **named layer** and a **user style** for it. The layer and style can have metadata elements such as a **name** identifying them, a **title** for displaying them, and an **abstract** describing them in detail. Within the top-level style are one or more **feature type styles**, which act as "virtual layers" to provide control over rendering order (allowing styling effects such as cased lines for roads). Each feature type style contains one or more **rules**, which control how styling is applied based on feature attributes and zoom level. Rules select applicable features by using **filters**, which are logical conditions containing **predicates**, **expressions** and **filter functions**. To specify the details of styling for individual features, rules contain any number of **symbolizers**. Symbolizers specify styling for **points**, **lines** and **polygons**, as well as **rasters** and **text labels**.

For more information refer to the *SLD Reference*.

Types of styling

Vector data that GeoServer can serve consists of three classes of shapes: **Points, lines, and polygons**. Lines (one dimensional shapes) are the simplest, as they have only the edge to style (also known as "stroke"). Polygons, two dimensional shapes, have an edge and an inside (also known as a "fill"), both of which can be styled differently. Points, even though they lack dimension, have both an edge and a fill (not to mention a size) that can be styled. For fills, color can be specified; for strokes, color and thickness can be specified.

GeoServer also serves raster data. This can be styled with a wide variety of control over color palette, opacity, contrast and other parameters.

More advanced styling is possible as well. Points can be specified with well-known shapes like circles, squares, stars, and even custom graphics or text. Lines can be styled with a dash styles and hashes. Polygons can be filled with a custom tiled graphics. Styling can be based on attributes in the data, so that certain features are styled differently. Text labels on features are possible as well. Styling can also be determined by zoom level, so that features are displayed in a way appropriate to their apparent size. The possibilities are vast.

A basic style example

A good way to learn about SLD is to study styling examples. The following is a simple SLD that can be applied to a layer that contains points, to style them as red circles with a size of 6 pixels. (This is the first example in the *Points* section of the *SLD Cookbook*.)

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <StyledLayerDescriptor version="1.0.0"
3      xsi:schemaLocation="http://www.opengis.net/sld StyledLayerDescriptor.xsd"
4      xmlns="http://www.opengis.net/sld"
5      xmlns:ogc="http://www.opengis.net/ogc"
6      xmlns:xlink="http://www.w3.org/1999/xlink"
7      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
8    <NamedLayer>
9      <Name>Simple point</Name>
10     <UserStyle>
11       <Title>GeoServer SLD Cook Book: Simple point</Title>
12       <FeatureTypeStyle>
13         <Rule>
14           <PointSymbolizer>
15             <Graphic>
16               <Mark>
17                 <WellKnownName>circle</WellKnownName>
18                 <Fill>
19                   <CssParameter name="fill">#FF0000</CssParameter>
20                 </Fill>
21               </Mark>
22               <Size>6</Size>
23             </Graphic>
24           </PointSymbolizer>
25         </Rule>
26       </FeatureTypeStyle>
27     </UserStyle>
28   </NamedLayer>
29 </StyledLayerDescriptor>
```

Although the example looks long, only a few lines are really important to understand. **Line 14** states that a "PointSymbolizer" is to be used to style data as points. **Lines 15-17** state that points are to be styled using a graphic shape specified by a "well known name", in this case a circle. SLD provides names for many shapes such as "square", "star", "triangle", etc. **Lines 18-20** specify the shape should be filled with a color of `#FF0000` (red). This is an RGB color code, written in hexadecimal, in the form of #RRGGBB. Finally, **line 22** specifies that the size of the shape is 6 pixels in width. The rest of the structure contains metadata about the style, such as a name identifying the style and a title for use in legends.

Note: In SLD documents some tags have prefixes, such as `ogc:`. This is because they are defined in **XML namespaces**. The top-level `StyledLayerDescriptor` tag (**lines 2-7**) specifies two XML namespaces, one called `xmlns`, and one called `xmlns:ogc`. The first namespace is the default for the document, so tags belonging to it do not need a prefix. Tags belonging to the second require the prefix `ogc:`. In fact, the namespace prefixes can be any identifier. The first namespace could be called `xmlns:sld` (as it often is) and then all the tags in this example would require an `sld:` prefix. The key point is that tags need to have the prefix for the namespace they belong to.

See the *SLD Cookbook* for more examples of styling with SLD.

6.2.2 Working with SLD

This section describes how to create, view and troubleshoot SLD styling in GeoServer.

Creating

GeoServer comes with some basic styles defined in its catalog. Any number of new styles can be added to the catalog. Styles can also be specified **externally** to the server, either to define a complete map, or to extend the server style catalog using **library mode**.

Catalog Styles

Styles in the catalog can be viewed, edited and validated via the *Styles page* menu of the *Web administration interface*. They may also be created and accessed via the REST *Styles* API.

There are two types of Catalog Styles: Symbology Encoding styles (the default) and Style Layer Descriptor styles.

Symbology Encoding Styles

A Symbology Encoding style consists of a `Symbology Encoding` document used to specify the styling of a single layer. In GeoServer, this is more commonly referred to as a style.

GeoServer supports the use of a *StyledLayerDescriptor* document containing a single `<NamedLayer>` element, which contains a single `<UserStyle>` element to specify the styling.

- When used in this fashion the layer name is ignored, since the style may be applied to many different layers.
- When using an an *StyledLayerDescriptor* generated by another application keep in mind only the first `<NamedLayer>` is used, any subsequent content is ignored.

Every layer (featuretype) registered with GeoServer must have at least one symbology encoding style associated with it, which is the default style for rendering the layer. Any number of additional styles can be associated with a layer. This allows layers to have appropriate styles advertised in the WMS `GetCapabilities` document. A layer's styles can be changed using the *Layers* page of the *Web administration interface*.

Note: When adding a layer and a style for it to GeoServer at the same time, the style should be added first, so that the new layer can be associated with the style immediately.

Style Layer Descriptor Styles

A Style Layer Descriptor is a *StyledLayerDescriptor* document containing any number of `<NamedLayer>` and `<UserLayer>` elements, each of which may contain any number of `<UserStyle>` or `<NamedStyle>` elements.

Within a Style Layer Descriptor document, the name of any `<NamedLayer>` elements should match a layer (or layer group) in the catalog. Likewise, any `<NamedStyle>` elements should refer to a style in the catalog.

Style Layer Descriptor styles can define new layers of styled data, by using the *InlineFeature* element to provide feature data.

Within GeoServer, when Style Layer Descriptor styles are used they are typically in the form of style groups. Style groups can be added to layer groups as an alternative way of defining a collection of styled layers, using either the *Web Administration interface* or the REST API.

Style Layer Descriptor styles can still be assigned to layers and used like a layer style, in which case only the first `<NamedLayer>` will be used. Style Layer Descriptor styles can also be used as an External Style, via the geoserver styles endpoint (`/geoserver/styles`) or the geoserver REST api.

External Styles

Styling can be defined externally to the server in a number of ways:

- An internet-accessible SLD document can be provided via the `SLD=url` parameter in a WMS *GetMap* GET request
- An SLD document can be provided directly in a WMS *GetMap* GET request using the `SLD_BODY=style` parameter. The SLD XML must be URL-encoded.
- A *StyledLayerDescriptor* element can be included in a WMS `GetMap` POST request XML document.

In all of these cases, if the WMS `layers` parameter is not supplied then the map content is defined completely by the layers and styles present in the external SLD. If the `layers` parameter is present, then styling operates in *Library Mode*.

The structure of an external style is the same as a Style Layer Descriptor style, as described above.

External styles can define new layers of styled data, by using the SLD *InlineFeature* element to provide feature data. This can be used to implement dynamic feature highlighting, for example.

External styling may be generated dynamically by client applications, This provides a powerful way for clients to control styling effects.

Library Mode

In **library mode** externally-defined styles are treated as a *style library*, which acts as an extension to the server style catalog. Library mode occurs when map layers and styles are specified using the `layers` and `styles` WMS parameters, and additional styling is supplied externally using one of the methods described in the previous section. The styles in the external style document take precedence over the catalog styles during rendering.

Style lookup in library mode operates as follows:

- For each layer in the `layers` list, the applied style is either a named style specified in the `styles` list (if present), or the layer default style
- For a **named** style, if the external style document has a `<NamedLayer>...<UserStyle>` with matching layer name and style name, then it is used. Otherwise, the style name is searched for in the catalog. If it is not found there, an error occurs.
- For a **default** style, the external style document is searched to find a `<NamedLayer>` element with the layer name. If it contains a `<UserStyle>` with the `<IsDefault>` element having the value `1` then that style is used. Otherwise, the default server style for the layer (which must exist) is used.

Generally it is simpler and more performant to use styles from the server catalog. However, library mode can be useful if it is required to style a map containing many layers and where only a few of them need to have their styling defined externally.

Viewing

Once a style has been associated with a layer, the resulting rendering of the layer data can be viewed by using the *Layer Preview*. The most convenient output format to use is the built-in OpenLayers viewer.

Styles can be modified while the view is open, and their effect is visible as soon as the map view is panned or zoomed. Alternate styles can be viewed by specifying them in the `styles` WMS request parameter.

To view the effect of compositing multiple styled layers, several approaches are available:

- Create a **layer group** for the desired layers using the *Layer Groups* page, and preview it. Non-default styles can be specified for layers if required.
- Submit a WMS *GetMap* GET request specifying multiple layers in the `layers` parameter, and the corresponding styles in the `styles` parameter (if non-default styles are required).
- Submit a WMS `GetMap` POST request containing a *StyledLayerDescriptor* element specifying server layers, optional layers of inline data, and either named catalog styles or user-defined styling for each layer.

Troubleshooting

SLD is a type of programming language, not unlike creating a web page or building a script. As such, problems may arise that may require troubleshooting.

Syntax Errors

To minimize syntax errors when creating the SLD, it is recommended to use a text editor that is designed to work with XML (such as the *Style Editor* provided in the GeoServer UI). XML editors can make finding syntax errors easier by providing syntax highlighting and (sometimes) built-in error checking.

The GeoServer *Style Editor* allows validating a document against the SLD XML schema. This is not mandatory, but is recommended to do before saving styles.

Semantic Errors

Semantic errors cannot be caught by SLD validation, but show up when a style is applied during map rendering. Most of the time this will result in a map displaying no features (a blank map), but some errors will prevent the map from rendering at all.

The easiest way to fix semantic errors in an SLD is to try to isolate the error. If the SLD is long with many rules and filters, try temporarily removing some of them to see if the errors go away.

In some cases the server will produce a WMS Exception document which may help to identify the error. It is also worth checking the server log to see if any error messages have been recorded.

6.2.3 SLD Cookbook

The SLD Cookbook is a collection of SLD "recipes" for creating various types of map styles. Wherever possible, each example is designed to show off a single SLD feature so that code can be copied from the examples and adapted when creating SLDs of your own. While not an exhaustive reference like the *SLD Reference* or the OGC SLD 1.0 specification the SLD Cookbook is designed to be a practical reference, showing common style templates that are easy to understand.

The SLD Cookbook is divided into four sections: the first three for each of the vector types (points, lines, and polygons) and the fourth section for rasters. Each example in every section contains a screenshot showing actual GeoServer WMS output, a snippet of the SLD code for reference, and a link to download the full SLD.

Each section uses data created especially for the SLD Cookbook, with shapefiles for vector data and GeoTIFFs for raster data. The projection for data is EPSG:4326. All files can be easily loaded into GeoServer in order to recreate the examples.

Data Type	Shapefile
Point	`sld_cookbook_point.zip`
Line	`sld_cookbook_line.zip`
Polygon	`sld_cookbook_polygon.zip`
Raster	`sld_cookbook_raster.zip`

Points

While points are seemingly the simplest type of shape, possessing only position and no other dimensions, there are many different ways that a point can be styled in SLD.

Warning: The code examples shown on this page are **not the full SLD code**, as they omit the SLD header and footer information for the sake of brevity. Please use the links to download the full SLD for each example.

Example points layer

The `points layer` used for the examples below contains name and population information for the major cities of a fictional country. For reference, the attribute table for the points in this layer is included below.

fid (Feature ID)	**name** (City name)	**pop** (Population)
point.1	Borfin	157860
point.2	Supox City	578231
point.3	Ruckis	98159
point.4	Thisland	34879
point.5	Synopolis	24567
point.6	San Glissando	76024
point.7	Detrania	205609

`Download the points shapefile`

Simple point

This example specifies points be styled as red circles with a diameter of 6 pixels.

Code

`View and download the full "Simple point" SLD`

```
<FeatureTypeStyle>
  <Rule>
    <PointSymbolizer>
      <Graphic>
        <Mark>
```

Fig. 6.20: *Simple point*

```
6             <WellKnownName>circle</WellKnownName>
7             <Fill>
8               <CssParameter name="fill">#FF0000</CssParameter>
9             </Fill>
10          </Mark>
11          <Size>6</Size>
12        </Graphic>
13      </PointSymbolizer>
14    </Rule>
15  </FeatureTypeStyle>
```

Details

There is one `<Rule>` in one `<FeatureTypeStyle>` for this SLD, which is the simplest possible situation. (All subsequent examples will contain one `<Rule>` and one `<FeatureTypeStyle>` unless otherwise specified.) Styling points is accomplished via the `<PointSymbolizer>` (**lines 3-13**). **Line 6** specifies the shape of the symbol to be a circle, with **line 8** determining the fill color to be red (`#FF0000`). **Line 11** sets the size (diameter) of the graphic to be 6 pixels.

Simple point with stroke

This example adds a stroke (or border) around the *Simple point*, with the stroke colored black and given a thickness of 2 pixels.

Code

`View and download the full "Simple point with stroke" SLD`

```
1   <FeatureTypeStyle>
2     <Rule>
3       <PointSymbolizer>
4         <Graphic>
5           <Mark>
6             <WellKnownName>circle</WellKnownName>
7             <Fill>
```

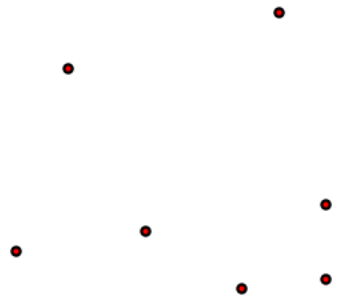


Fig. 6.21: *Simple point with stroke*

```
8               <CssParameter name="fill">#FF0000</CssParameter>
9             </Fill>
10            <Stroke>
11              <CssParameter name="stroke">#000000</CssParameter>
12              <CssParameter name="stroke-width">2</CssParameter>
13            </Stroke>
14          </Mark>
15          <Size>6</Size>
16        </Graphic>
17      </PointSymbolizer>
18    </Rule>
19  </FeatureTypeStyle>
```

Details

This example is similar to the *Simple point* example. **Lines 10-13** specify the stroke, with **line 11** setting the color to black (`#000000`) and **line 12** setting the width to 2 pixels.

Rotated square

This example creates a square instead of a circle, colors it green, sizes it to 12 pixels, and rotates it by 45 degrees.

Code

View and download the full "Rotated square" SLD

```
1   <FeatureTypeStyle>
2     <Rule>
3       <PointSymbolizer>
4         <Graphic>
5           <Mark>
6             <WellKnownName>square</WellKnownName>
7             <Fill>
8               <CssParameter name="fill">#009900</CssParameter>
```

Fig. 6.22: *Rotated square*

```
9               </Fill>
10            </Mark>
11            <Size>12</Size>
12            <Rotation>45</Rotation>
13          </Graphic>
14        </PointSymbolizer>
15      </Rule>
16    </FeatureTypeStyle>
```

Details

In this example, **line 6** sets the shape to be a square, with **line 8** setting the color to a dark green (`#009900`). **Line 11** sets the size of the square to be 12 pixels, and **line 12** set the rotation is to 45 degrees.

Transparent triangle

This example draws a triangle, creates a black stroke identical to the *Simple point with stroke* example, and sets the fill of the triangle to 20% opacity (mostly transparent).

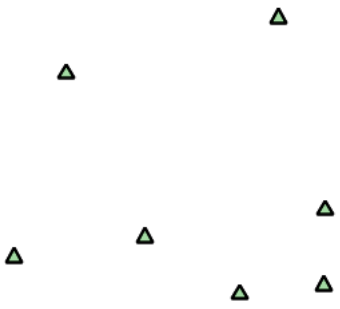


Fig. 6.23: *Transparent triangle*

Code

View and download the full "Transparent triangle" SLD

```
1  <FeatureTypeStyle>
2    <Rule>
3      <PointSymbolizer>
4        <Graphic>
5          <Mark>
6            <WellKnownName>triangle</WellKnownName>
7            <Fill>
8              <CssParameter name="fill">#009900</CssParameter>
9              <CssParameter name="fill-opacity">0.2</CssParameter>
10           </Fill>
11           <Stroke>
12             <CssParameter name="stroke">#000000</CssParameter>
13             <CssParameter name="stroke-width">2</CssParameter>
14           </Stroke>
15         </Mark>
16         <Size>12</Size>
17       </Graphic>
18     </PointSymbolizer>
19   </Rule>
20 </FeatureTypeStyle>
```

Details

In this example, **line 6** once again sets the shape, in this case to a triangle. **Line 8** sets the fill color to a dark green (`#009900`) and **line 9** sets the opacity to 0.2 (20% opaque). An opacity value of 1 means that the shape is drawn 100% opaque, while an opacity value of 0 means that the shape is drawn 0% opaque, or completely transparent. The value of 0.2 (20% opaque) means that the fill of the points partially takes on the color and style of whatever is drawn beneath it. In this example, since the background is white, the dark green looks lighter. Were the points imposed on a dark background, the resulting color would be darker. **Lines 12-13** set the stroke color to black (`#000000`) and width to 2 pixels. Finally, **line 16** sets the size of the point to be 12 pixels in diameter.

Point as graphic

This example styles each point as a graphic instead of as a simple shape.

Code

View and download the full "Point as graphic" SLD

```
1  <FeatureTypeStyle>
2    <Rule>
3      <PointSymbolizer>
4        <Graphic>
5          <ExternalGraphic>
6            <OnlineResource
7              xlink:type="simple"
8              xlink:href="smileyface.png" />
9            <Format>image/png</Format>
```

Fig. 6.24: *Point as graphic*

```
10            </ExternalGraphic>
11            <Size>32</Size>
12          </Graphic>
13        </PointSymbolizer>
14      </Rule>
15    </FeatureTypeStyle>
```

Details

This style uses a graphic instead of a simple shape to render the points. In SLD, this is known as an `<ExternalGraphic>`, to distinguish it from the commonly-used shapes such as squares and circles that are "internal" to the renderer. **Lines 5-10** specify the details of this graphic. **Line 8** sets the path and file name of the graphic, while **line 9** indicates the format (MIME type) of the graphic (image/png). In this example, the graphic is contained in the same directory as the SLD, so no path information is necessary in **line 8**, although a full URL could be used if desired. **Line 11** determines the size of the displayed graphic; this can be set independently of the dimensions of the graphic itself, although in this case they are the same (32 pixels). Should a graphic be rectangular, the `<Size>` value will apply to the *height* of the graphic only, with the width scaled proportionally.

Fig. 6.25: *Graphic used for points*

Point with default label

This example shows a text label on the *Simple point* that displays the "name" attribute of the point. This is how a label will be displayed in the absence of any other customization.

Code

`View and download the full "Point with default label" SLD`

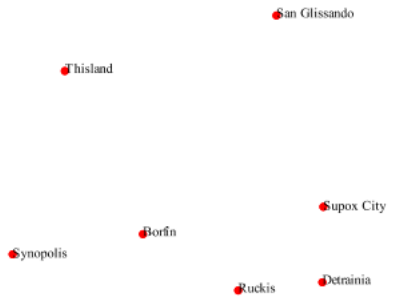


Fig. 6.26: *Point with default label*

```
1   <FeatureTypeStyle>
2     <Rule>
3       <PointSymbolizer>
4         <Graphic>
5           <Mark>
6             <WellKnownName>circle</WellKnownName>
7             <Fill>
8               <CssParameter name="fill">#FF0000</CssParameter>
9             </Fill>
10          </Mark>
11          <Size>6</Size>
12        </Graphic>
13      </PointSymbolizer>
14      <TextSymbolizer>
15        <Label>
16          <ogc:PropertyName>name</ogc:PropertyName>
17        </Label>
18        <Fill>
19          <CssParameter name="fill">#000000</CssParameter>
20        </Fill>
21      </TextSymbolizer>
22    </Rule>
23  </FeatureTypeStyle>
```

Details

Lines 3-13, which contain the `<PointSymbolizer>`, are identical to the *Simple point* example above. The label is set in the `<TextSymbolizer>` on **lines 14-27**. **Lines 15-17** determine what text to display in the label, which in this case is the value of the "name" attribute. (Refer to the attribute table in the *Example points layer* section if necessary.) **Line 19** sets the text color. All other details about the label are set to the renderer default, which here is Times New Roman font, font color black, and font size of 10 pixels. The bottom left of the label is aligned with the center of the point.

Point with styled label

This example improves the label style from the *Point with default label* example by centering the label above the point and providing a different font name and size.

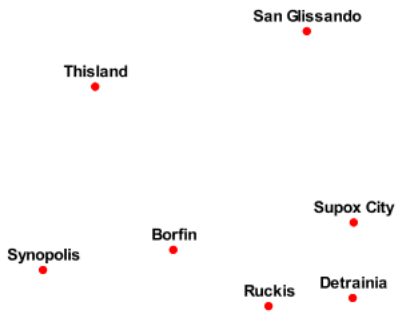


Fig. 6.27: *Point with styled label*

Code

View and download the full "Point with styled label" SLD

```
<FeatureTypeStyle>
  <Rule>
    <PointSymbolizer>
      <Graphic>
        <Mark>
          <WellKnownName>circle</WellKnownName>
          <Fill>
            <CssParameter name="fill">#FF0000</CssParameter>
          </Fill>
        </Mark>
        <Size>6</Size>
      </Graphic>
    </PointSymbolizer>
    <TextSymbolizer>
      <Label>
        <ogc:PropertyName>name</ogc:PropertyName>
      </Label>
      <Font>
        <CssParameter name="font-family">Arial</CssParameter>
        <CssParameter name="font-size">12</CssParameter>
        <CssParameter name="font-style">normal</CssParameter>
        <CssParameter name="font-weight">bold</CssParameter>
      </Font>
      <LabelPlacement>
        <PointPlacement>
          <AnchorPoint>
            <AnchorPointX>0.5</AnchorPointX>
            <AnchorPointY>0.0</AnchorPointY>
          </AnchorPoint>
          <Displacement>
            <DisplacementX>0</DisplacementX>
            <DisplacementY>5</DisplacementY>
          </Displacement>
        </PointPlacement>
      </LabelPlacement>
      <Fill>
```

```
37            <CssParameter name="fill">#000000</CssParameter>
38          </Fill>
39        </TextSymbolizer>
40      </Rule>
41    </FeatureTypeStyle>
```

Details

In this example, **lines 3-13** are identical to the *Simple point* example above. The `<TextSymbolizer>` on lines 14-39 contains many more details about the label styling than the previous example, *Point with default label*. **Lines 15-17** once again specify the "name" attribute as text to display. **Lines 18-23** set the font information: **line 19** sets the font family to be "Arial", **line 20** sets the font size to 12, **line 21** sets the font style to "normal" (as opposed to "italic" or "oblique"), and **line 22** sets the font weight to "bold" (as opposed to "normal"). **Lines 24-35** (`<LabelPlacement>`) determine the placement of the label relative to the point. The `<AnchorPoint>` (**lines 26-29**) sets the point of intersection between the label and point, which here (**line 27-28**) sets the point to be centered (0.5) horizontally axis and bottom aligned (0.0) vertically with the label. There is also `<Displacement>` (**lines 30-33**), which sets the offset of the label relative to the line, which in this case is 0 pixels horizontally (**line 31**) and 5 pixels vertically (**line 32**). Finally, **line 37** sets the font color of the label to black (`#000000`).

The result is a centered bold label placed slightly above each point.

Point with rotated label

This example builds on the previous example, *Point with styled label*, by rotating the label by 45 degrees, positioning the labels farther away from the points, and changing the color of the label to purple.

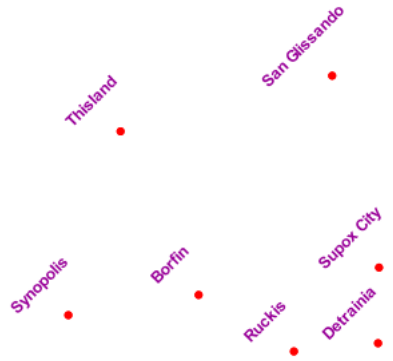


Fig. 6.28: *Point with rotated label*

Code

View and download the full "Point with rotated label" SLD

```
1   <FeatureTypeStyle>
2     <Rule>
3       <PointSymbolizer>
4         <Graphic>
```

```
5           <Mark>
6             <WellKnownName>circle</WellKnownName>
7             <Fill>
8               <CssParameter name="fill">#FF0000</CssParameter>
9             </Fill>
10          </Mark>
11          <Size>6</Size>
12        </Graphic>
13      </PointSymbolizer>
14      <TextSymbolizer>
15        <Label>
16          <ogc:PropertyName>name</ogc:PropertyName>
17        </Label>
18        <Font>
19          <CssParameter name="font-family">Arial</CssParameter>
20          <CssParameter name="font-size">12</CssParameter>
21          <CssParameter name="font-style">normal</CssParameter>
22          <CssParameter name="font-weight">bold</CssParameter>
23        </Font>
24        <LabelPlacement>
25          <PointPlacement>
26            <AnchorPoint>
27              <AnchorPointX>0.5</AnchorPointX>
28              <AnchorPointY>0.0</AnchorPointY>
29            </AnchorPoint>
30            <Displacement>
31              <DisplacementX>0</DisplacementX>
32              <DisplacementY>25</DisplacementY>
33            </Displacement>
34            <Rotation>-45</Rotation>
35          </PointPlacement>
36        </LabelPlacement>
37        <Fill>
38          <CssParameter name="fill">#990099</CssParameter>
39        </Fill>
40      </TextSymbolizer>
41    </Rule>
42  </FeatureTypeStyle>
```

Details

This example is similar to the *Point with styled label*, but there are three important differences. **Line 32** specifies 25 pixels of vertical displacement. **Line 34** specifies a rotation of "-45" or 45 degrees counter-clockwise. (Rotation values increase clockwise, which is why the value is negative.) Finally, **line 38** sets the font color to be a shade of purple (`#990099`).

Note that the displacement takes effect before the rotation during rendering, so in this example, the 25 pixel vertical displacement is itself rotated 45 degrees.

Attribute-based point

This example alters the size of the symbol based on the value of the population ("pop") attribute.

Fig. 6.29: *Attribute-based point*

Code

View and download the full "Attribute-based point" SLD

```
<FeatureTypeStyle>
  <Rule>
    <Name>SmallPop</Name>
    <Title>1 to 50000</Title>
    <ogc:Filter>
      <ogc:PropertyIsLessThan>
        <ogc:PropertyName>pop</ogc:PropertyName>
        <ogc:Literal>50000</ogc:Literal>
      </ogc:PropertyIsLessThan>
    </ogc:Filter>
    <PointSymbolizer>
      <Graphic>
        <Mark>
          <WellKnownName>circle</WellKnownName>
          <Fill>
            <CssParameter name="fill">#0033CC</CssParameter>
          </Fill>
        </Mark>
        <Size>8</Size>
      </Graphic>
    </PointSymbolizer>
  </Rule>
  <Rule>
    <Name>MediumPop</Name>
    <Title>50000 to 100000</Title>
    <ogc:Filter>
      <ogc:And>
        <ogc:PropertyIsGreaterThanOrEqualTo>
          <ogc:PropertyName>pop</ogc:PropertyName>
          <ogc:Literal>50000</ogc:Literal>
        </ogc:PropertyIsGreaterThanOrEqualTo>
        <ogc:PropertyIsLessThan>
          <ogc:PropertyName>pop</ogc:PropertyName>
          <ogc:Literal>100000</ogc:Literal>
        </ogc:PropertyIsLessThan>
      </ogc:And>
```

```
      </ogc:Filter>
      <PointSymbolizer>
        <Graphic>
          <Mark>
            <WellKnownName>circle</WellKnownName>
            <Fill>
              <CssParameter name="fill">#0033CC</CssParameter>
            </Fill>
          </Mark>
          <Size>12</Size>
        </Graphic>
      </PointSymbolizer>
    </Rule>
    <Rule>
      <Name>LargePop</Name>
      <Title>Greater than 100000</Title>
      <ogc:Filter>
        <ogc:PropertyIsGreaterThanOrEqualTo>
          <ogc:PropertyName>pop</ogc:PropertyName>
          <ogc:Literal>100000</ogc:Literal>
        </ogc:PropertyIsGreaterThanOrEqualTo>
      </ogc:Filter>
      <PointSymbolizer>
        <Graphic>
          <Mark>
            <WellKnownName>circle</WellKnownName>
            <Fill>
              <CssParameter name="fill">#0033CC</CssParameter>
            </Fill>
          </Mark>
          <Size>16</Size>
        </Graphic>
      </PointSymbolizer>
    </Rule>
  </FeatureTypeStyle>
```

Details

Note: Refer to the *Example points layer* to see the attributes for this data. This example has eschewed labels in order to simplify the style, but you can refer to the example *Point with styled label* to see which attributes correspond to which points.

This style contains three rules. Each `<Rule>` varies the style based on the value of the population ("pop") attribute for each point, with smaller values yielding a smaller circle, and larger values yielding a larger circle.

The three rules are designed as follows:

Rule order	Rule name	Population ("pop")	Size
1	SmallPop	Less than 50,000	8
2	MediumPop	50,000 to 100,000	12
3	LargePop	Greater than 100,000	16

The order of the rules does not matter in this case, since each shape is only rendered by a single rule.

The first rule, on **lines 2-22**, specifies the styling of those points whose population attribute is less than 50,000. **Lines 5-10** set this filter, with **lines 6-9** setting the "less than" filter, **line 7** denoting the attribute ("pop"), and **line 8** the value of 50,000. The symbol is a circle (**line 14**), the color is dark blue (`#0033CC`, on **line 16**), and the size is 8 pixels in diameter (**line 19**).

The second rule, on **lines 23-49**, specifies a style for points whose population attribute is greater than or equal to 50,000 and less than 100,000. The population filter is set on **lines 26-37**. This filter is longer than in the first rule because two criteria need to be specified instead of one: a "greater than or equal to" and a "less than" filter. Notice the `And` on **line 27** and **line 36**. This mandates that both filters need to be true for the rule to be applicable. The size of the graphic is set to 12 pixels on **line 46**. All other styling directives are identical to the first rule.

The third rule, on **lines 50-70**, specifies a style for points whose population attribute is greater than or equal to 100,000. The population filter is set on **lines 53-58**, and the only other difference is the size of the circle, which in this rule (**line 67**) is 16 pixels.

The result of this style is that cities with larger populations have larger points.

Zoom-based point

This example alters the style of the points at different zoom levels.

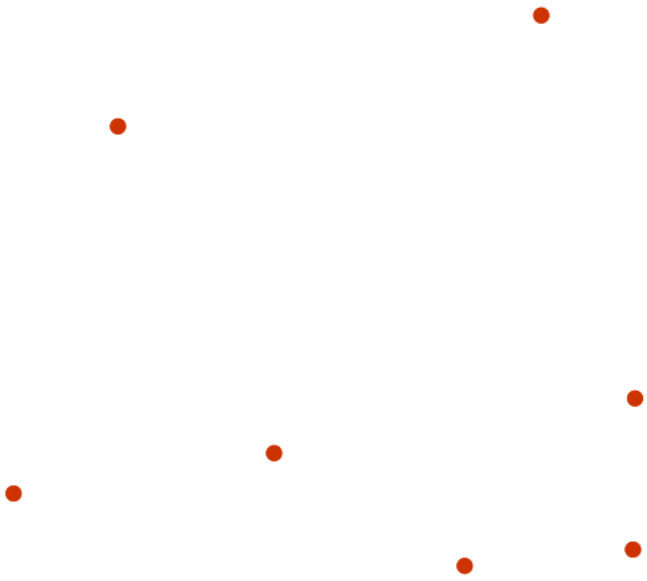


Fig. 6.30: *Zoom-based point: Zoomed in*

Code

`View and download the full "Zoom-based point" SLD`

Fig. 6.31: *Zoom-based point: Partially zoomed*

Fig. 6.32: *Zoom-based point: Zoomed out*

```
<FeatureTypeStyle>
  <Rule>
    <Name>Large</Name>
    <MaxScaleDenominator>160000000</MaxScaleDenominator>
    <PointSymbolizer>
      <Graphic>
        <Mark>
          <WellKnownName>circle</WellKnownName>
          <Fill>
            <CssParameter name="fill">#CC3300</CssParameter>
          </Fill>
        </Mark>
        <Size>12</Size>
      </Graphic>
    </PointSymbolizer>
  </Rule>
  <Rule>
    <Name>Medium</Name>
    <MinScaleDenominator>160000000</MinScaleDenominator>
    <MaxScaleDenominator>320000000</MaxScaleDenominator>
    <PointSymbolizer>
      <Graphic>
        <Mark>
          <WellKnownName>circle</WellKnownName>
          <Fill>
            <CssParameter name="fill">#CC3300</CssParameter>
          </Fill>
        </Mark>
        <Size>8</Size>
      </Graphic>
    </PointSymbolizer>
  </Rule>
  <Rule>
    <Name>Small</Name>
    <MinScaleDenominator>320000000</MinScaleDenominator>
    <PointSymbolizer>
      <Graphic>
        <Mark>
          <WellKnownName>circle</WellKnownName>
          <Fill>
            <CssParameter name="fill">#CC3300</CssParameter>
          </Fill>
        </Mark>
        <Size>4</Size>
      </Graphic>
    </PointSymbolizer>
  </Rule>
</FeatureTypeStyle>
```

Details

It is often desirable to make shapes larger at higher zoom levels when creating a natural-looking map. This example styles the points to vary in size based on the zoom level (or more accurately, scale denominator). Scale denominators refer to the scale of the map. A scale denominator of 10,000 means the map has a scale of 1:10,000 in the units of the map projection.

Note: Determining the appropriate scale denominators (zoom levels) to use is beyond the scope of this example.

This style contains three rules. The three rules are designed as follows:

Rule order	Rule name	Scale denominator	Point size
1	Large	1:160,000,000 or less	12
2	Medium	1:160,000,000 to 1:320,000,000	8
3	Small	Greater than 1:320,000,000	4

The order of these rules does not matter since the scales denominated in each rule do not overlap.

The first rule (**lines 2-16**) is for the smallest scale denominator, corresponding to when the view is "zoomed in". The scale rule is set on **line 4**, so that the rule will apply to any map with a scale denominator of 160,000,000 or less. The rule draws a circle (**line 8**), colored red (`#CC3300` on **line 10**) with a size of 12 pixels (**line 13**).

The second rule (**lines 17-32**) is the intermediate scale denominator, corresponding to when the view is "partially zoomed". The scale rules are set on **lines 19-20**, so that the rule will apply to any map with a scale denominator between 160,000,000 and 320,000,000. (The `<MinScaleDenominator>` is inclusive and the `<MaxScaleDenominator>` is exclusive, so a zoom level of exactly 320,000,000 would *not* apply here.) Aside from the scale, the only difference between this rule and the first is the size of the symbol, which is set to 8 pixels on **line 29**.

The third rule (**lines 33-47**) is the largest scale denominator, corresponding to when the map is "zoomed out". The scale rule is set on **line 35**, so that the rule will apply to any map with a scale denominator of 320,000,000 or more. Again, the only other difference between this rule and the others is the size of the symbol, which is set to 4 pixels on **line 44**.

The result of this style is that points are drawn larger as one zooms in and smaller as one zooms out.

Lines

While lines can also seem to be simple shapes, having length but no width, there are many options and tricks for making lines display nicely.

Warning: The code examples shown on this page are **not the full SLD code**, as they omit the SLD header and footer information for the sake of brevity. Please use the links to download the full SLD for each example.

Example lines layer

The `lines layer` used in the examples below contains road information for a fictional country. For reference, the attribute table for the points in this layer is included below.

fid (Feature ID)	**name** (Road name)	**type** (Road class)
line.1	Latway	highway
line.2	Crescent Avenue	secondary
line.3	Forest Avenue	secondary
line.4	Longway	highway
line.5	Saxer Avenue	secondary
line.6	Ridge Avenue	secondary
line.7	Holly Lane	local-road
line.8	Mulberry Street	local-road
line.9	Nathan Lane	local-road
line.10	Central Street	local-road
line.11	Lois Lane	local-road
line.12	Rocky Road	local-road
line.13	Fleet Street	local-road
line.14	Diane Court	local-road
line.15	Cedar Trail	local-road
line.16	Victory Road	local-road
line.17	Highland Road	local-road
line.18	Easy Street	local-road
line.19	Hill Street	local-road
line.20	Country Road	local-road
line.21	Main Street	local-road
line.22	Jani Lane	local-road
line.23	Shinbone Alley	local-road
line.24	State Street	local-road
line.25	River Road	local-road

Download the lines shapefile

Simple line

This example specifies lines be colored black with a thickness of 3 pixels.

Code

View and download the full "Simple line" SLD

```
  <FeatureTypeStyle>
    <Rule>
      <LineSymbolizer>
        <Stroke>
          <CssParameter name="stroke">#000000</CssParameter>
          <CssParameter name="stroke-width">3</CssParameter>
        </Stroke>
      </LineSymbolizer>
    </Rule>
  </FeatureTypeStyle>
```

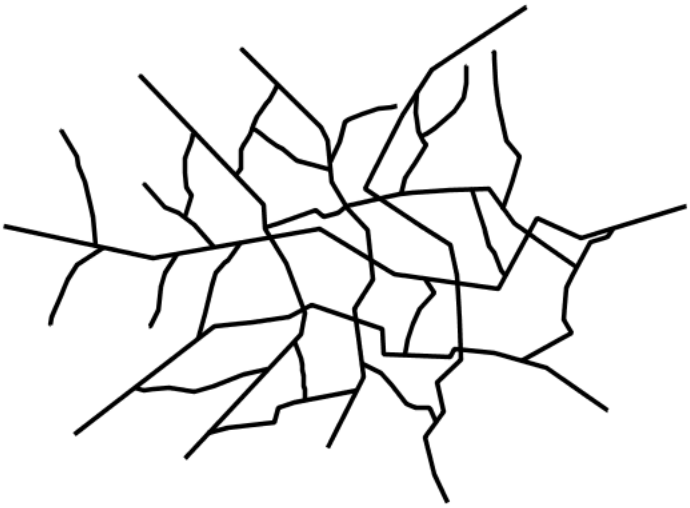


Fig. 6.33: *Simple line*

Details

There is one `<Rule>` in one `<FeatureTypeStyle>` for this SLD, which is the simplest possible situation. (All subsequent examples will contain one `<Rule>` and one `<FeatureTypeStyle>` unless otherwise specified.) Styling lines is accomplished via the `<LineSymbolizer>` (**lines 3-8**). **Line 5** specifies the color of the line to be black (`#000000`), while **line 6** specifies the width of the lines to be 3 pixels.

Line with border

This example shows how to draw lines with borders (sometimes called "cased lines"). In this case the lines are drawn with a 3 pixel blue center and a 1 pixel wide gray border.

Code

`View and download the full "Line with border" SLD`

```
1    <FeatureTypeStyle>
2       <Rule>
3        <LineSymbolizer>
4          <Stroke>
5            <CssParameter name="stroke">#333333</CssParameter>
6            <CssParameter name="stroke-width">5</CssParameter>
7            <CssParameter name="stroke-linecap">round</CssParameter>
8          </Stroke>
9        </LineSymbolizer>
10     </Rule>
11   </FeatureTypeStyle>
12   <FeatureTypeStyle>
13      <Rule>
14       <LineSymbolizer>
```

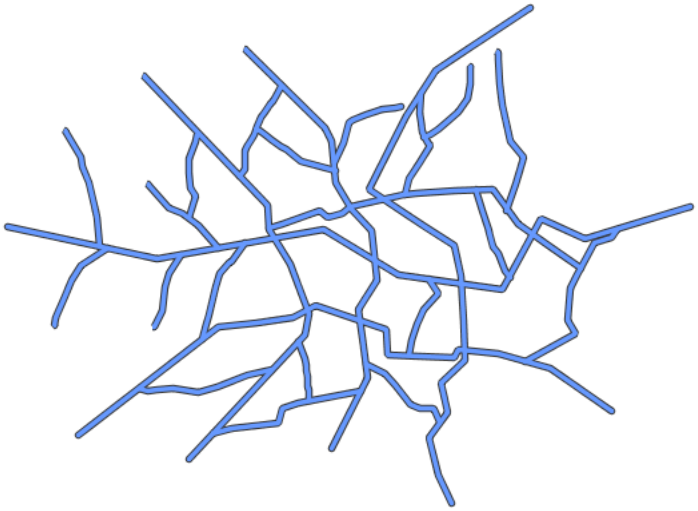


Fig. 6.34: *Line with border*

```
15        <Stroke>
16            <CssParameter name="stroke">#6699FF</CssParameter>
17            <CssParameter name="stroke-width">3</CssParameter>
18            <CssParameter name="stroke-linecap">round</CssParameter>
19          </Stroke>
20        </LineSymbolizer>
21      </Rule>
22    </FeatureTypeStyle>
```

Details

Lines in SLD have no notion of a "fill", only "stroke". Thus, unlike points or polygons, it is not possible to style the "edge" of the line geometry. It is, however, possible to achieve this effect by drawing each line twice: once with a certain width and again with a slightly smaller width. This gives the illusion of fill and stroke by obscuring the larger lines everywhere except along the edges of the smaller lines.

Since every line is drawn twice, the order of the rendering is *very* important. GeoServer renders `<FeatureTypeStyle>`s in the order that they are presented in the SLD. In this style, the gray border lines are drawn first via the first `<FeatureTypeStyle>`, followed by the blue center lines in a second `<FeatureTypeStyle>`. This ensures that the blue lines are not obscured by the gray lines, and also ensures proper rendering at intersections, so that the blue lines "connect".

In this example, **lines 1-11** comprise the first `<FeatureTypeStyle>`, which is the outer line (or "stroke"). **Line 5** specifies the color of the line to be dark gray (`#333333`), **line 6** specifies the width of this line to be 5 pixels, and in **line 7** a `stroke-linecap` parameter of `round` renders the ends of the line as rounded instead of flat. (When working with bordered lines using a round line cap ensures that the border connects properly at the ends of the lines.)

Lines 12-22 comprise the second `<FeatureTypeStyle>`, which is the the inner line (or "fill"). **Line 16** specifies the color of the line to be a medium blue (`#6699FF`), **line 17** specifies the width of this line to be 3 pixels, and **line 18** again renders the edges of the line to be rounded instead of flat.

The result is a 3 pixel blue line with a 1 pixel gray border, since the 5 pixel gray line will display 1 pixel on each side of the 3 pixel blue line.

Dashed line

This example alters the *Simple line* to create a dashed line consisting of 5 pixels of drawn line alternating with 2 pixels of blank space.

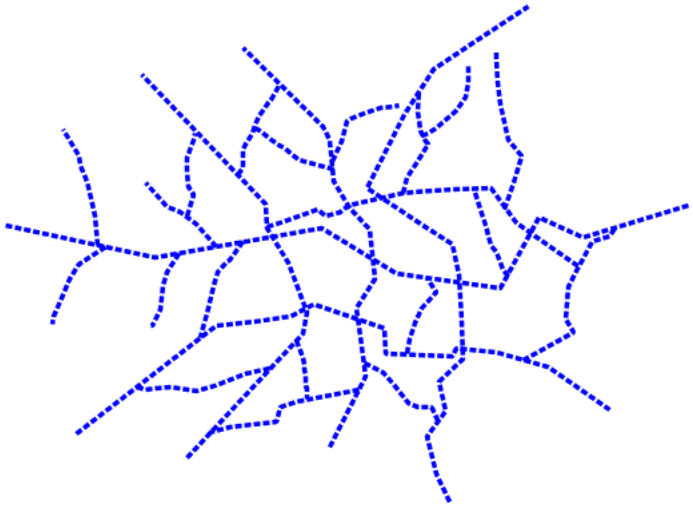


Fig. 6.35: *Dashed line*

Code

View and download the full "Dashed line" SLD

```
1   <FeatureTypeStyle>
2     <Rule>
3       <LineSymbolizer>
4         <Stroke>
5           <CssParameter name="stroke">#0000FF</CssParameter>
6           <CssParameter name="stroke-width">3</CssParameter>
7           <CssParameter name="stroke-dasharray">5 2</CssParameter>
8         </Stroke>
9       </LineSymbolizer>
10    </Rule>
11  </FeatureTypeStyle>
```

Details

In this example, **line 5** sets the color of the lines to be blue (`#0000FF`) and **line 6** sets the width of the lines to be 3 pixels. **Line 7** determines the composition of the line dashes. The value of `5 2` creates a repeating pattern of 5 pixels of drawn line, followed by 2 pixels of omitted line.

Offset line

This example alters the *Simple line* to add a perpendicular offset line on the left side of the line, at five pixels distance.

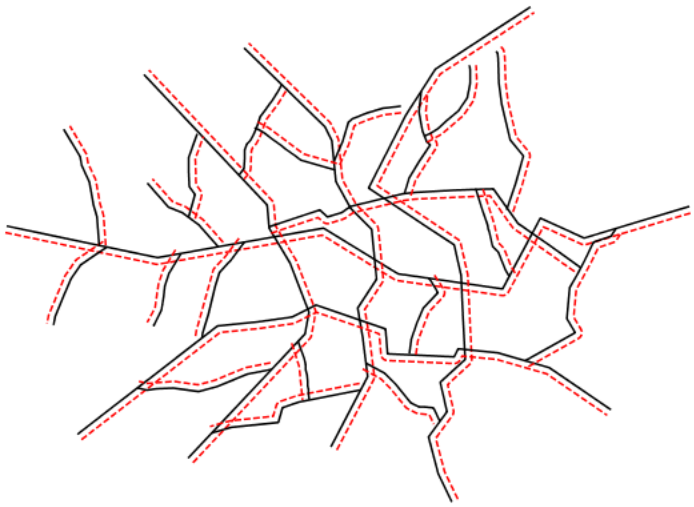


Fig. 6.36: *Offset line*

Code

View and download the full "Dashed line" SLD

```
1   <FeatureTypeStyle>
2     <Rule>
3       <LineSymbolizer>
4         <Stroke>
5           <CssParameter name="stroke">#000000</CssParameter>
6         </Stroke>
7       </LineSymbolizer>
8       <LineSymbolizer>
9         <Stroke>
10          <CssParameter name="stroke">#FF0000</CssParameter>
11          <CssParameter name="stroke-dasharray">5 2</CssParameter>
12        </Stroke>
13        <PerpendicularOffset>5</PerpendicularOffset>
14      </LineSymbolizer>
15    </Rule>
16  </FeatureTypeStyle>
```

Details

In this example, the first line symbolizer just paints the lines black. **line 8** begines a second lines symbolizer, sets the color of the lines to be red (`#FF0000`) at line 10 and determines the composition of the line dashes

at **Line 11**. **Line 13** finally specifies a perpendicular offset of 5 pixels (positive, thus on the left side).

Railroad (hatching)

This example uses hatching to create a railroad style. Both the line and the hatches are black, with a 2 pixel thickness for the main line and a 1 pixel width for the perpendicular hatches.

Note: This example leverages an SLD extension in GeoServer. Hatching is not part of the standard SLD 1.0 specification.

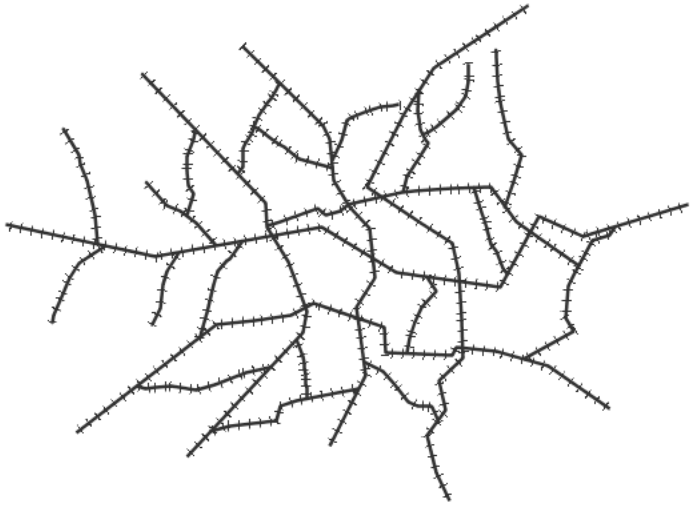


Fig. 6.37: *Railroad (hatching)*

Code

View and download the full "Railroad (hatching)" SLD

```
<FeatureTypeStyle>
  <Rule>
    <LineSymbolizer>
      <Stroke>
        <CssParameter name="stroke">#333333</CssParameter>
        <CssParameter name="stroke-width">3</CssParameter>
      </Stroke>
    </LineSymbolizer>
    <LineSymbolizer>
      <Stroke>
        <GraphicStroke>
          <Graphic>
            <Mark>
              <WellKnownName>shape://vertline</WellKnownName>
              <Stroke>
```

```
                    <CssParameter name="stroke">#333333</CssParameter>
                    <CssParameter name="stroke-width">1</CssParameter>
                  </Stroke>
                </Mark>
                <Size>12</Size>
              </Graphic>
            </GraphicStroke>
          </Stroke>
        </LineSymbolizer>
      </Rule>
    </FeatureTypeStyle>
```

Details

In this example there are two `<LineSymbolizer>`s. The first symbolizer, on **lines 3-8**, draws a standard line, with **line 5** drawing the lines as dark gray (`#333333`) and **line 6** setting the width of the lines to be 2 pixels.

The hatching is invoked in the second symbolizer, on **lines 9-24**. **Line 14** specifies that the symbolizer draw a vertical line hatch (`shape://vertline`) perpendicular to the line geometry. **Lines 16-17** set the hatch color to dark gray (`#333333`) and width to 1 pixel. Finally, **line 20** specifies both the length of the hatch and the distance between each hatch to both be 12 pixels.

Spaced graphic symbols

This example uses a graphic stroke along with dash arrays to create a "dot and space" line type. Adding the dash array specification allows to control the amount of space between one symbol and the next one. Without using the dash array the lines would be densely populated with dots, each one touching the previous one.

Note: This example may not work in other systems using SLD, since they may not support combining the use of `stroke-dasharray` and `GraphicStroke`. While the SLD is spec-compliant, the SLD specification does not state what this combination is supposed to produce.

Code

`View and download the full "Spaced symbols" SLD`

```
  <FeatureTypeStyle>
    <Rule>
      <LineSymbolizer>
        <Stroke>
          <GraphicStroke>
            <Graphic>
              <Mark>
                <WellKnownName>circle</WellKnownName>
                <Fill>
                  <CssParameter name="fill">#666666</CssParameter>
                </Fill>
                <Stroke>
                  <CssParameter name="stroke">#333333</CssParameter>
```

Fig. 6.38: *Spaced symbols along a line*

```
14                    <CssParameter name="stroke-width">1</CssParameter>
15                  </Stroke>
16                </Mark>
17                <Size>4</Size>
18              </Graphic>
19            </GraphicStroke>
20            <CssParameter name="stroke-dasharray">4 6</CssParameter>
21          </Stroke>
22        </LineSymbolizer>
23      </Rule>
24    </FeatureTypeStyle>
```

Details

This example, like others before, uses a `GraphicStroke` to place a graphic symbol along a line. The symbol, defined at **lines 7-16** is a 4 pixel gray circle with a dark gray outline. The spacing between symbols is controlled with the `stroke-dasharray` at **line 20**, which specifies 4 pixels of pen-down (just enough to draw the circle) and 6 pixels of pen-up, to provide the spacing.

Alternating symbols with dash offsets

This example shows how to create a complex line style which alternates a dashed line and a graphic symbol. The code builds on features shown in the previous examples:

- `stroke-dasharray` controls pen-down/pen-up behavior to generate dashed lines
- `GraphicStroke` places symbols along a line
- combining the two allows control of symbol spacing

This also shows the usage of a *dash offset*, which controls where rendering starts in the dash array. For example, with a dash array of `5 10` and a dash offset of `7` the renderer starts drawing the pattern 7 pixels from the beginning. It skips the 5 pixels pen-down section and 2 pixels of the pen-up section, then draws the remaining 8 pixels of pen-up, then 5 down, 10 up, and so on.

The example shows how to use these features to create two synchronized sequences of dash arrays, one drawing line segments and the other symbols.

Note: This example may not work in other systems using SLD, since they may not support combining the use of `stroke-dasharray` and `GraphicStroke`. While the SLD is spec-compliant, the SLD specification does not state what this combination is supposed to produce.

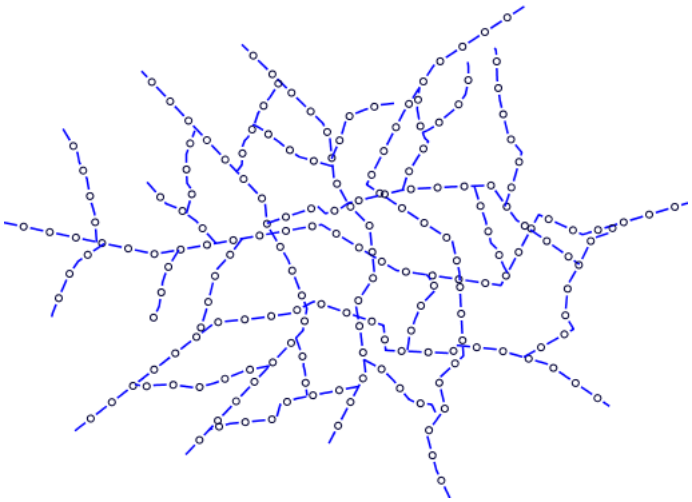


Fig. 6.39: *Alternating dash and symbol*

Code

View and download the full "Spaced symbols" SLD

```
  <FeatureTypeStyle>
    <Rule>
      <LineSymbolizer>
        <Stroke>
          <CssParameter name="stroke">#0000FF</CssParameter>
          <CssParameter name="stroke-width">1</CssParameter>
          <CssParameter name="stroke-dasharray">10 10</CssParameter>
        </Stroke>
      </LineSymbolizer>
      <LineSymbolizer>
        <Stroke>
          <GraphicStroke>
            <Graphic>
              <Mark>
                <WellKnownName>circle</WellKnownName>
                <Stroke>
                  <CssParameter name="stroke">#000033</CssParameter>
                  <CssParameter name="stroke-width">1</CssParameter>
                </Stroke>
              </Mark>
              <Size>5</Size>
            </Graphic>
          </GraphicStroke>
          <CssParameter name="stroke-dasharray">5 15</CssParameter>
```

```
25            <CssParameter name="stroke-dashoffset">7.5</CssParameter>
26          </Stroke>
27        </LineSymbolizer>
28      </Rule>
29    </FeatureTypeStyle>
```

Details

In this example two `LineSymbolizers` use `stroke-dasharray` and different symbology to produce a sequence of alternating dashes and symbols. The first symbolizer (**lines 3-9**) is a simple dashed line alternating 10 pixels of pen-down with 10 pixels of pen-up. The second symbolizer (**lines 10-27**) alternates a 5 pixel empty circle with 15 pixels of white space. The circle symbol is produced by a `Mark` element, with its symbology specified by `stroke` parameters (**lines 17-18**). The spacing between symbols is controlled with the `stroke-dasharray` (**line 24**), which specifies 5 pixels of pen-down (just enough to draw the circle) and 15 pixels of pen-up. In order to have the two sequences positioned correctly the second one uses a `stroke-dashoffset` of 7.5 (**line 25**). This makes the sequence start with 12.5 pixels of white space, then a circle (which is then centered between the two line segments of the other pattern), then 15 pixels of white space, and so on.

Line with default label

This example shows a text label on the simple line. This is how a label will be displayed in the absence of any other customization.

Fig. 6.40: *Line with default label*

Code

`View and download the full "Line with default label" SLD`

```
1  <FeatureTypeStyle>
2    <Rule>
3      <LineSymbolizer>
4        <Stroke>
5          <CssParameter name="stroke">#FF0000</CssParameter>
6        </Stroke>
7      </LineSymbolizer>
8      <TextSymbolizer>
9        <Label>
10         <ogc:PropertyName>name</ogc:PropertyName>
11       </Label>
12       <LabelPlacement>
13         <LinePlacement />
14       </LabelPlacement>
15       <Fill>
16         <CssParameter name="fill">#000000</CssParameter>
17       </Fill>
18     </TextSymbolizer>
19   </Rule>
20 </FeatureTypeStyle>
```

Details

In this example, there is one rule with a `<LineSymbolizer>` and a `<TextSymbolizer>`. The `<LineSymbolizer>` (**lines 3-7**) draws red lines (`#FF0000`). Since no width is specified, the default is set to 1 pixel. The `<TextSymbolizer>` (**lines 8-15**) determines the labeling of the lines. **Lines 9-11** specify that the text of the label will be determined by the value of the "name" attribute for each line. (Refer to the attribute table in the *Example lines layer* section if necessary.) **Line 13** sets the text color to black. All other details about the label are set to the renderer default, which here is Times New Roman font, font color black, and font size of 10 pixels.

Label following line

This example renders the text label to follow the contour of the lines.

Note: Labels following lines is an SLD extension specific to GeoServer. It is not part of the SLD 1.0 specification.

Code

`View and download the full "Label following line" SLD`

```
1  <FeatureTypeStyle>
2    <Rule>
3      <LineSymbolizer>
4        <Stroke>
5          <CssParameter name="stroke">#FF0000</CssParameter>
6        </Stroke>
7      </LineSymbolizer>
8      <TextSymbolizer>
9        <Label>
```

Fig. 6.41: *Label following line*

```
10            <ogc:PropertyName>name</ogc:PropertyName>
11          </Label>
12          <LabelPlacement>
13            <LinePlacement />
14          </LabelPlacement>
15          <Fill>
16            <CssParameter name="fill">#000000</CssParameter>
17          </Fill>
18          <VendorOption name="followLine">true</VendorOption>
19        </TextSymbolizer>
20      </Rule>
21    </FeatureTypeStyle>
```

Details

As the *Alternating symbols with dash offsets* example showed, the default label behavior isn't optimal. The label is displayed at a tangent to the line itself, leading to uncertainty as to which label corresponds to which line.

This example is similar to the *Alternating symbols with dash offsets* example with the exception of **lines 12-18**. **Line 18** sets the option to have the label follow the line, while **lines 12-14** specify that the label is placed along a line. If `<LinePlacement />` is not specified in an SLD, then `<PointPlacement />` is assumed, which isn't compatible with line-specific rendering options.

Note: Not all labels are shown due to label conflict resolution. See the next section on *Optimized label placement* for an example of how to maximize label display.

Optimized label placement

This example optimizes label placement for lines such that the maximum number of labels are displayed.

Note: This example uses options that are specific to GeoServer and are not part of the SLD 1.0 specification.

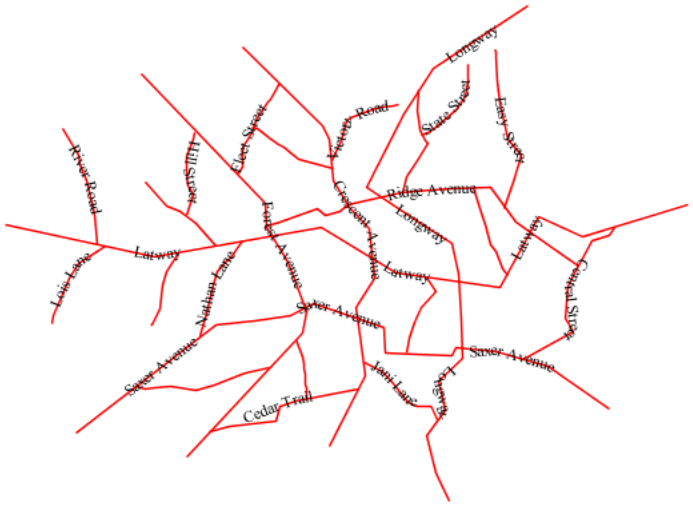


Fig. 6.42: *Optimized label*

Code

View and download the full "Optimized label" SLD

```
  <FeatureTypeStyle>
    <Rule>
      <LineSymbolizer>
        <Stroke>
          <CssParameter name="stroke">#FF0000</CssParameter>
        </Stroke>
      </LineSymbolizer>
      <TextSymbolizer>
        <Label>
          <ogc:PropertyName>name</ogc:PropertyName>
        </Label>
        <LabelPlacement>
           <LinePlacement />
        </LabelPlacement>
        <Fill>
          <CssParameter name="fill">#000000</CssParameter>
        </Fill>
        <VendorOption name="followLine">true</VendorOption>
        <VendorOption name="maxAngleDelta">90</VendorOption>
        <VendorOption name="maxDisplacement">400</VendorOption>
        <VendorOption name="repeat">150</VendorOption>
```

```
      </TextSymbolizer>
    </Rule>
  </FeatureTypeStyle>
```

Details

GeoServer uses "conflict resolution" to ensure that labels aren't drawn on top of other labels, obscuring them both. This accounts for the reason why many lines don't have labels in the previous example, *Label following line*. While this setting can be toggled, it is usually a good idea to leave it on and use other label placement options to ensure that labels are drawn as often as desired and in the correct places. This example does just that.

This example is similar to the previous example, *Label following line*. The only differences are contained in **lines 18-21**. **Line 19** sets the maximum angle that the label will follow. This sets the label to never bend more than 90 degrees to prevent the label from becoming illegible due to a pronounced curve or angle. **Line 20** sets the maximum displacement of the label to be 400 pixels. In order to resolve conflicts with overlapping labels, GeoServer will attempt to move the labels such that they are no longer overlapping. This value sets how far the label can be moved relative to its original placement. Finally, **line 21** sets the labels to be repeated every 150 pixels. A feature will typically receive only one label, but this can cause confusion for long lines. Setting the label to repeat ensures that the line is always labeled locally.

Optimized and styled label

This example improves the style of the labels from the *Optimized label placement* example.

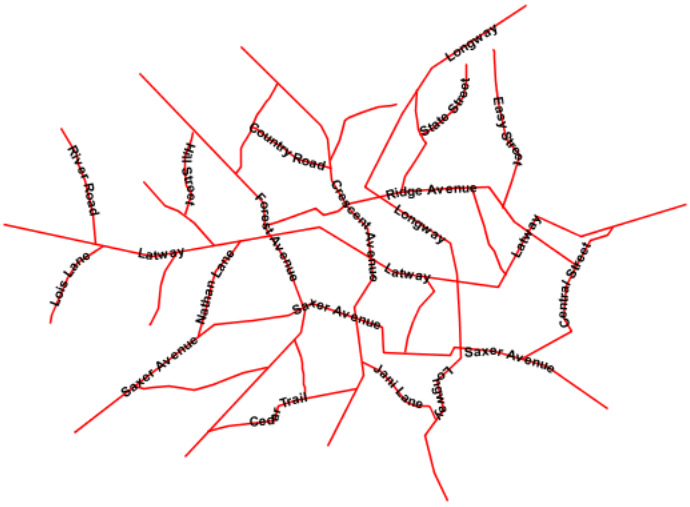


Fig. 6.43: *Optimized and styled label*

Code

View and download the full "Optimized and styled label" SLD

```
  <FeatureTypeStyle>
    <Rule>
      <LineSymbolizer>
        <Stroke>
          <CssParameter name="stroke">#FF0000</CssParameter>
        </Stroke>
      </LineSymbolizer>
      <TextSymbolizer>
        <Label>
          <ogc:PropertyName>name</ogc:PropertyName>
        </Label>
        <LabelPlacement>
          <LinePlacement />
        </LabelPlacement>
        <Fill>
          <CssParameter name="fill">#000000</CssParameter>
        </Fill>
        <Font>
          <CssParameter name="font-family">Arial</CssParameter>
          <CssParameter name="font-size">10</CssParameter>
          <CssParameter name="font-style">normal</CssParameter>
          <CssParameter name="font-weight">bold</CssParameter>
        </Font>
        <VendorOption name="followLine">true</VendorOption>
        <VendorOption name="maxAngleDelta">90</VendorOption>
        <VendorOption name="maxDisplacement">400</VendorOption>
        <VendorOption name="repeat">150</VendorOption>
      </TextSymbolizer>
    </Rule>
  </FeatureTypeStyle>
```

Details

This example is similar to the *Optimized label placement*. The only difference is in the font information, which is contained in **lines 18-23**. **Line 19** sets the font family to be "Arial", **line 20** sets the font size to 10, **line 21** sets the font style to "normal" (as opposed to "italic" or "oblique"), and **line 22** sets the font weight to "bold" (as opposed to "normal").

Attribute-based line

This example styles the lines differently based on the "type" (Road class) attribute.

Code

View and download the full "Attribute-based line" SLD

```
  <FeatureTypeStyle>
    <Rule>
      <Name>local-road</Name>
      <ogc:Filter>
        <ogc:PropertyIsEqualTo>
          <ogc:PropertyName>type</ogc:PropertyName>
          <ogc:Literal>local-road</ogc:Literal>
```

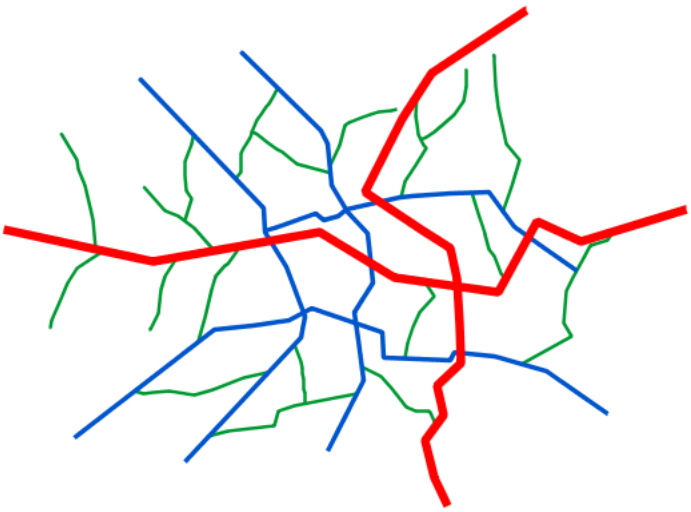


Fig. 6.44: *Attribute-based line*

```
        </ogc:PropertyIsEqualTo>
      </ogc:Filter>
      <LineSymbolizer>
        <Stroke>
          <CssParameter name="stroke">#009933</CssParameter>
          <CssParameter name="stroke-width">2</CssParameter>
        </Stroke>
      </LineSymbolizer>
    </Rule>
  </FeatureTypeStyle>
  <FeatureTypeStyle>
    <Rule>
      <Name>secondary</Name>
      <ogc:Filter>
        <ogc:PropertyIsEqualTo>
          <ogc:PropertyName>type</ogc:PropertyName>
          <ogc:Literal>secondary</ogc:Literal>
        </ogc:PropertyIsEqualTo>
      </ogc:Filter>
      <LineSymbolizer>
        <Stroke>
          <CssParameter name="stroke">#0055CC</CssParameter>
          <CssParameter name="stroke-width">3</CssParameter>
        </Stroke>
      </LineSymbolizer>
    </Rule>
  </FeatureTypeStyle>
  <FeatureTypeStyle>
    <Rule>
    <Name>highway</Name>
      <ogc:Filter>
        <ogc:PropertyIsEqualTo>
          <ogc:PropertyName>type</ogc:PropertyName>
```

```
41            <ogc:Literal>highway</ogc:Literal>
42          </ogc:PropertyIsEqualTo>
43        </ogc:Filter>
44        <LineSymbolizer>
45          <Stroke>
46            <CssParameter name="stroke">#FF0000</CssParameter>
47            <CssParameter name="stroke-width">6</CssParameter>
48          </Stroke>
49        </LineSymbolizer>
50      </Rule>
51    </FeatureTypeStyle>
```

Details

Note: Refer to the *Example lines layer* to see the attributes for the layer. This example has eschewed labels in order to simplify the style, but you can refer to the example *Optimized and styled label* to see which attributes correspond to which points.

There are three types of road classes in our fictional country, ranging from back roads to high-speed freeways: "highway", "secondary", and "local-road". In order to handle each case separately, there is more than one `<FeatureTypeStyle>`, each containing a single rule. This ensures that each road type is rendered in order, as each `<FeatureTypeStyle>` is drawn based on the order in which it appears in the SLD.

The three rules are designed as follows:

Rule order	Rule name / type	Color	Size
1	local-road	`#009933` (green)	2
2	secondary	`#0055CC` (blue)	3
3	highway	`#FF0000` (red)	6

Lines 2-16 comprise the first `<Rule>`. **Lines 4-9** set the filter for this rule, such that the "type" attribute has a value of "local-road". If this condition is true for a particular line, the rule is rendered according to the `<LineSymbolizer>` which is on **lines 10-15**. **Lines 12-13** set the color of the line to be a dark green (`#009933`) and the width to be 2 pixels.

Lines 19-33 comprise the second `<Rule>`. **Lines 21-26** set the filter for this rule, such that the "type" attribute has a value of "secondary". If this condition is true for a particular line, the rule is rendered according to the `<LineSymbolizer>` which is on **lines 27-32**. **Lines 29-30** set the color of the line to be a dark blue (`#0055CC`) and the width to be 3 pixels, making the lines slightly thicker than the "local-road" lines and also a different color.

Lines 36-50 comprise the third and final `<Rule>`. **Lines 38-43** set the filter for this rule, such that the "type" attribute has a value of "primary". If this condition is true for a particular line, the rule is rendered according to the `<LineSymbolizer>` which is on **lines 44-49**. **Lines 46-47** set the color of the line to be a bright red (`#FF0000`) and the width to be 6 pixels, so that these lines are rendered on top of and thicker than the other two road classes. In this way, the "primary" roads are given priority in the map rendering.

Zoom-based line

This example alters the *Simple line* style at different zoom levels.

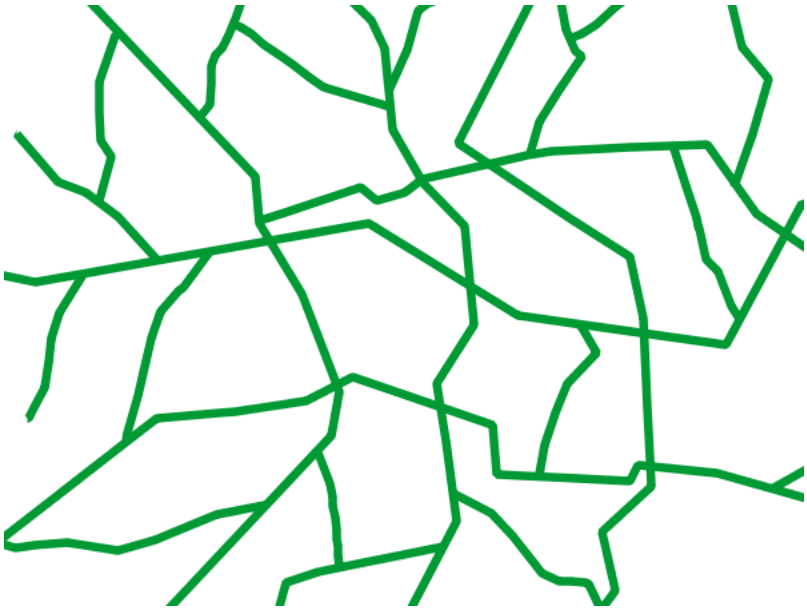


Fig. 6.45: *Zoom-based line: Zoomed in*

Fig. 6.46: *Zoom-based line: Partially zoomed*

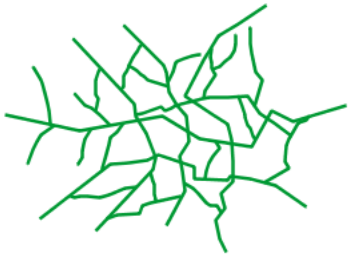


Fig. 6.47: *Zoom-based line: Zoomed out*

Code

View and download the full "Zoom-based line" SLD

```
  <FeatureTypeStyle>
    <Rule>
      <Name>Large</Name>
      <MaxScaleDenominator>180000000</MaxScaleDenominator>
      <LineSymbolizer>
        <Stroke>
          <CssParameter name="stroke">#009933</CssParameter>
          <CssParameter name="stroke-width">6</CssParameter>
        </Stroke>
      </LineSymbolizer>
    </Rule>
    <Rule>
      <Name>Medium</Name>
      <MinScaleDenominator>180000000</MinScaleDenominator>
      <MaxScaleDenominator>360000000</MaxScaleDenominator>
      <LineSymbolizer>
        <Stroke>
          <CssParameter name="stroke">#009933</CssParameter>
          <CssParameter name="stroke-width">4</CssParameter>
        </Stroke>
      </LineSymbolizer>
    </Rule>
    <Rule>
      <Name>Small</Name>
      <MinScaleDenominator>360000000</MinScaleDenominator>
      <LineSymbolizer>
        <Stroke>
          <CssParameter name="stroke">#009933</CssParameter>
          <CssParameter name="stroke-width">2</CssParameter>
        </Stroke>
```

```
      </LineSymbolizer>
    </Rule>
  </FeatureTypeStyle>
```

Details

It is often desirable to make shapes larger at higher zoom levels when creating a natural-looking map. This example varies the thickness of the lines according to the zoom level (or more accurately, scale denominator). Scale denominators refer to the scale of the map. A scale denominator of 10,000 means the map has a scale of 1:10,000 in the units of the map projection.

Note: Determining the appropriate scale denominators (zoom levels) to use is beyond the scope of this example.

This style contains three rules. The three rules are designed as follows:

Rule order	Rule name	Scale denominator	Line width
1	Large	1:180,000,000 or less	6
2	Medium	1:180,000,000 to 1:360,000,000	4
3	Small	Greater than 1:360,000,000	2

The order of these rules does not matter since the scales denominated in each rule do not overlap.

The first rule (**lines 2-11**) is the smallest scale denominator, corresponding to when the view is "zoomed in". The scale rule is set on **line 4**, so that the rule will apply to any map with a scale denominator of 180,000,000 or less. **Line 7-8** draws the line to be dark green (`#009933`) with a width of 6 pixels.

The second rule (**lines 12-22**) is the intermediate scale denominator, corresponding to when the view is "partially zoomed". **Lines 14-15** set the scale such that the rule will apply to any map with scale denominators between 180,000,000 and 360,000,000. (The `<MinScaleDenominator>` is inclusive and the `<MaxScaleDenominator>` is exclusive, so a zoom level of exactly 360,000,000 would *not* apply here.) Aside from the scale, the only difference between this rule and the previous is the width of the lines, which is set to 4 pixels on **line 19**.

The third rule (**lines 23-32**) is the largest scale denominator, corresponding to when the map is "zoomed out". The scale rule is set on **line 25**, so that the rule will apply to any map with a scale denominator of 360,000,000 or greater. Again, the only other difference between this rule and the others is the width of the lines, which is set to 2 pixels on **line 29**.

The result of this style is that lines are drawn with larger widths as one zooms in and smaller widths as one zooms out.

Polygons

Polygons are two dimensional shapes that contain both an outer edge (or "stroke") and an inside (or "fill"). A polygon can be thought of as an irregularly-shaped point and is styled in similar ways to points.

Warning: The code examples shown on this page are **not the full SLD code**, as they omit the SLD header and footer information for the sake of brevity. Please use the links to download the full SLD for each example.

Example polygons layer

The `polygons layer` used below contains county information for a fictional country. For reference, the attribute table for the polygons is included below.

fid (Feature ID)	**name** (County name)	**pop** (Population)
polygon.1	Irony County	412234
polygon.2	Tracker County	235421
polygon.3	Dracula County	135022
polygon.4	Poly County	1567879
polygon.5	Bearing County	201989
polygon.6	Monte Cristo County	152734
polygon.7	Massive County	67123
polygon.8	Rhombus County	198029

`Download the polygons shapefile`

Simple polygon

This example shows a polygon filled in blue.

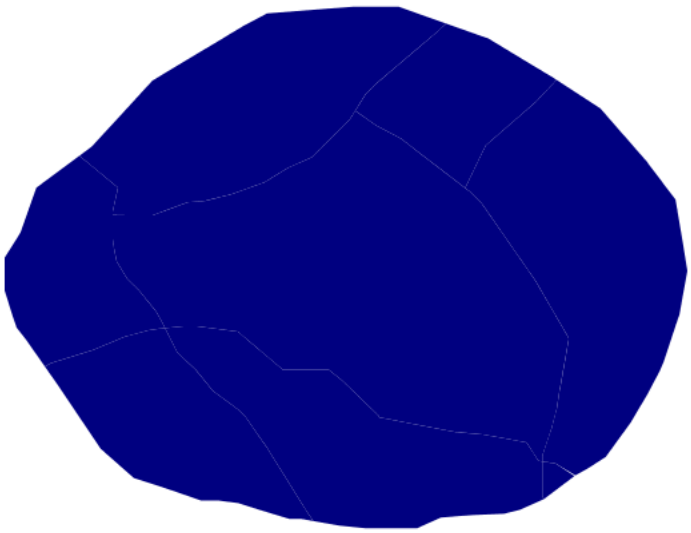


Fig. 6.48: *Simple polygon*

Code

`View and download the full "Simple polygon" SLD`

```
<FeatureTypeStyle>
  <Rule>
    <PolygonSymbolizer>
```

```
4           <Fill>
5             <CssParameter name="fill">#000080</CssParameter>
6           </Fill>
7         </PolygonSymbolizer>
8       </Rule>
9     </FeatureTypeStyle>
```

Details

There is one `<Rule>` in one `<FeatureTypeStyle>` for this style, which is the simplest possible situation. (All subsequent examples will share this characteristic unless otherwise specified.) Styling polygons is accomplished via the `<PolygonSymbolizer>` (**lines 3-7**). **Line 5** specifies dark blue (`#000080`) as the polygon's fill color.

Note: The light-colored borders around the polygons in the figure are artifacts of the renderer caused by the polygons being adjacent. There is no border in this style.

Simple polygon with stroke

This example adds a 2 pixel white stroke to the *Simple polygon* example.

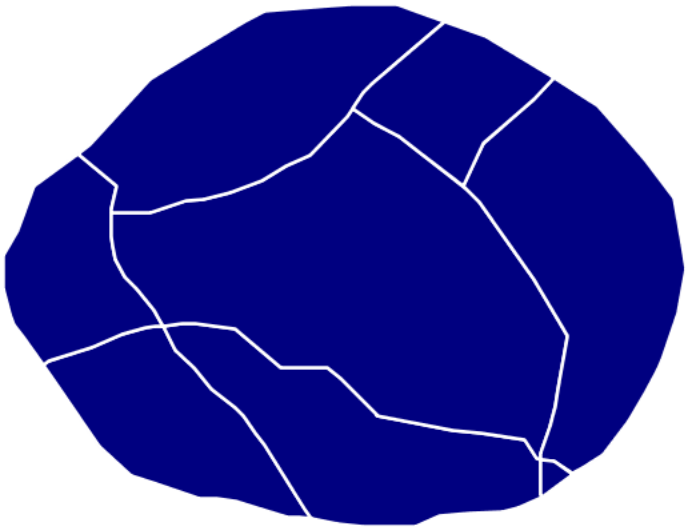


Fig. 6.49: *Simple polygon with stroke*

Code

`View and download the full "Simple polygon with stroke" SLD`

```
1  <FeatureTypeStyle>
2    <Rule>
3      <PolygonSymbolizer>
4        <Fill>
5          <CssParameter name="fill">#000080</CssParameter>
6        </Fill>
7        <Stroke>
8          <CssParameter name="stroke">#FFFFFF</CssParameter>
9          <CssParameter name="stroke-width">2</CssParameter>
10       </Stroke>
11     </PolygonSymbolizer>
12   </Rule>
13 </FeatureTypeStyle>
```

Details

This example is similar to the *Simple polygon* example above, with the addition of the `<Stroke>` tag (**lines 7-10**). **Line 8** sets the color of stroke to white (`#FFFFFF`) and **line 9** sets the width of the stroke to 2 pixels.

Transparent polygon

This example builds on the *Simple polygon with stroke* example and makes the fill partially transparent by setting the opacity to 50%.

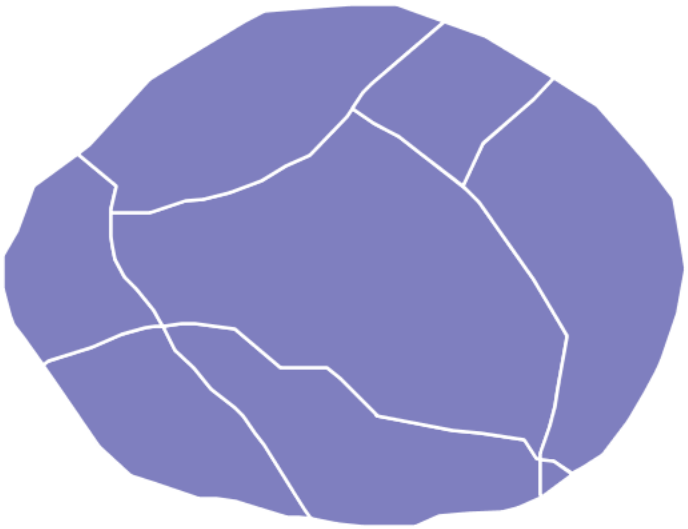


Fig. 6.50: *Transparent polygon*

Code

`View and download the full "Transparent polygon" SLD`

```
1   <FeatureTypeStyle>
2     <Rule>
3       <PolygonSymbolizer>
4         <Fill>
5           <CssParameter name="fill">#000080</CssParameter>
6           <CssParameter name="fill-opacity">0.5</CssParameter>
7         </Fill>
8         <Stroke>
9           <CssParameter name="stroke">#FFFFFF</CssParameter>
10          <CssParameter name="stroke-width">2</CssParameter>
11        </Stroke>
12      </PolygonSymbolizer>
13    </Rule>
14  </FeatureTypeStyle>
```

Details

This example is similar to the *Simple polygon with stroke* example, save for defining the fill's opacity in **line 6**. The value of 0.5 results in partially transparent fill that is 50% opaque. An opacity value of 1 would draw the fill as 100% opaque, while an opacity value of 0 would result in a completely transparent (0% opaque) fill. In this example, since the background is white, the dark blue looks lighter. Were the points imposed on a dark background, the resulting color would be darker.

Offset inner lines

Shows how to draw inner buffer lines inside a polygon.

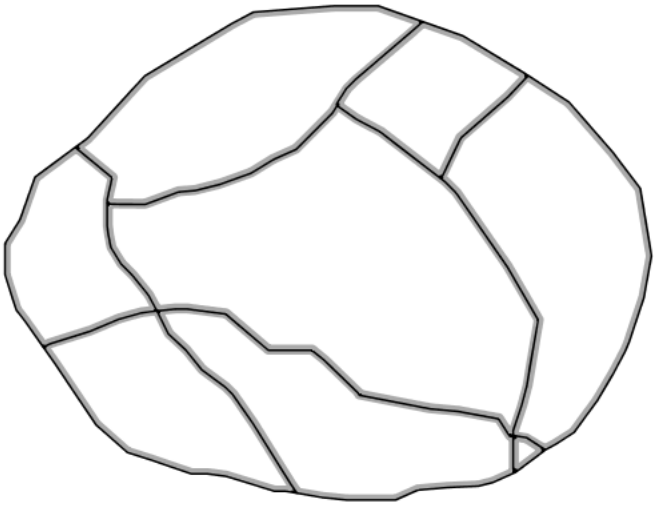


Fig. 6.51: *Offset buffer*

Code

View and download the full "Inner offset lines" SLD

```
1   <FeatureTypeStyle>
2     <Rule>
3       <PolygonSymbolizer>
4         <Stroke>
5           <CssParameter name="stroke">#000000</CssParameter>
6           <CssParameter name="stroke-width">2</CssParameter>
7         </Stroke>
8       </PolygonSymbolizer>
9       <LineSymbolizer>
10        <Stroke>
11          <CssParameter name="stroke">#AAAAAA</CssParameter>
12          <CssParameter name="stroke-width">3</CssParameter>
13        </Stroke>
14        <PerpendicularOffset>-2</PerpendicularOffset>
15      </LineSymbolizer>
16    </Rule>
17  </FeatureTypeStyle>
```

Details

This example is similar to the *Simple polygon with stroke* example, save for defining adding a `<LineSymbolizer>>` at **line 9**, where a light gray (**line 11**) 3 pixels wide (**line 12**) line is drawn as a inner buffer inside the polygon. **Line 14** controls the buffering distance, setting a inner buffer of 2 pixels.

Graphic fill

This example fills the polygons with a tiled graphic.

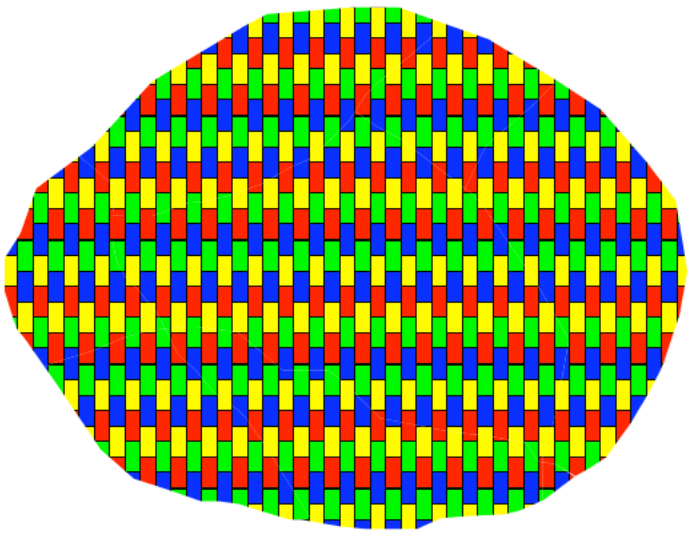


Fig. 6.52: *Graphic fill*

Code

View and download the full "Graphic fill" SLD

```
1  <FeatureTypeStyle>
2    <Rule>
3      <PolygonSymbolizer>
4        <Fill>
5          <GraphicFill>
6            <Graphic>
7              <ExternalGraphic>
8                <OnlineResource
9                  xlink:type="simple"
10                 xlink:href="colorblocks.png" />
11               <Format>image/png</Format>
12             </ExternalGraphic>
13           <Size>93</Size>
14           </Graphic>
15         </GraphicFill>
16       </Fill>
17     </PolygonSymbolizer>
18   </Rule>
19 </FeatureTypeStyle>
```

Details

This style fills the polygon with a tiled graphic. This is known as an `<ExternalGraphic>` in SLD, to distinguish it from commonly-used shapes such as squares and circles that are "internal" to the renderer. **Lines 7-12** specify details for the graphic, with **line 10** setting the path and file name of the graphic and **line 11** indicating the file format (MIME type) of the graphic (`image/png`). Although a full URL could be specified if desired, no path information is necessary in **line 11** because this graphic is contained in the same directory as the SLD. **Line 13** determines the height of the displayed graphic in pixels; if the value differs from the height of the graphic then it will be scaled accordingly while preserving the aspect ratio.

Fig. 6.53: *Graphic used for fill*

Hatching fill

This example fills the polygons with a hatching pattern.

Note: This example leverages an SLD extension in GeoServer. Hatching is not part of the standard SLD 1.0 specification.

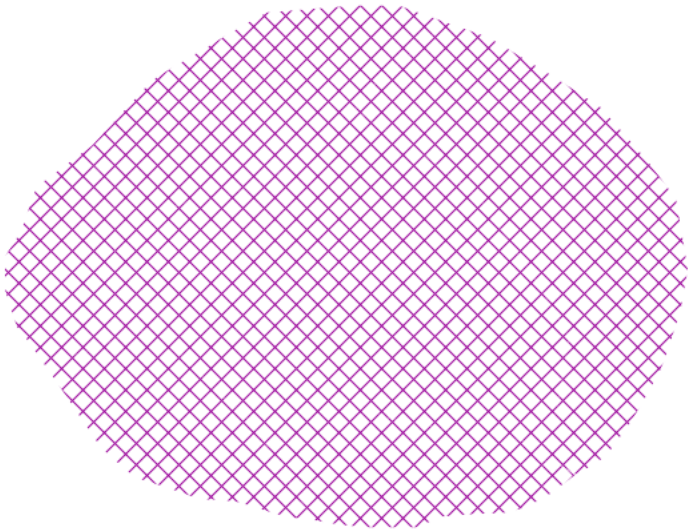


Fig. 6.54: *Hatching fill*

Code

View and download the full "Hatching fill" SLD

```
<FeatureTypeStyle>
  <Rule>
    <PolygonSymbolizer>
      <Fill>
        <GraphicFill>
          <Graphic>
            <Mark>
              <WellKnownName>shape://times</WellKnownName>
              <Stroke>
                <CssParameter name="stroke">#990099</CssParameter>
                <CssParameter name="stroke-width">1</CssParameter>
              </Stroke>
            </Mark>
            <Size>16</Size>
          </Graphic>
        </GraphicFill>
      </Fill>
    </PolygonSymbolizer>
  </Rule>
</FeatureTypeStyle>
```

Details

In this example, there is a `<GraphicFill>` tag as in the *Graphic fill* example, but a `<Mark>` (**lines 7-13**) is used instead of an `<ExternalGraphic>`. **Line 8** specifies a "times" symbol (an "x") be tiled throughout the polygon. **Line 10** sets the color to purple (`#990099`), **line 11** sets the width of the hatches to 1 pixel, and **line 14** sets the size of the tile to 16 pixels. Because hatch tiles are always square, the `<Size>` sets both the

width and the height.

Polygon with default label

This example shows a text label on the polygon. In the absence of any other customization, this is how a label will be displayed.

Fig. 6.55: *Polygon with default label*

Code

View and download the full "Polygon with default label" SLD

```
<FeatureTypeStyle>
  <Rule>
    <PolygonSymbolizer>
      <Fill>
        <CssParameter name="fill">#40FF40</CssParameter>
      </Fill>
      <Stroke>
        <CssParameter name="stroke">#FFFFFF</CssParameter>
        <CssParameter name="stroke-width">2</CssParameter>
      </Stroke>
    </PolygonSymbolizer>
    <TextSymbolizer>
      <Label>
        <ogc:PropertyName>name</ogc:PropertyName>
      </Label>
    </TextSymbolizer>
  </Rule>
</FeatureTypeStyle>
```

Details

In this example there is a `<PolygonSymbolizer>` and a `<TextSymbolizer>`. **Lines 3-11** comprise the `<PolygonSymbolizer>`. The fill of the polygon is set on **line 5** to a light green (`#40FF40`) while the stroke of the polygon is set on **lines 8-9** to white (`#FFFFFF`) with a thickness of 2 pixels. The label is set in the `<TextSymbolizer>` on **lines 12-16**, with **line 14** determining what text to display, in this case the value of the "name" attribute. (Refer to the attribute table in the *Example polygons layer* section if necessary.) All other details about the label are set to the renderer default, which here is Times New Roman font, font color black, and font size of 10 pixels.

Label halo

This example alters the look of the *Polygon with default label* by adding a white halo to the label.

Fig. 6.56: *Label halo*

Code

View and download the full "Label halo" SLD

```
1   <FeatureTypeStyle>
2     <Rule>
3       <PolygonSymbolizer>
4         <Fill>
5           <CssParameter name="fill">#40FF40</CssParameter>
6         </Fill>
7         <Stroke>
8           <CssParameter name="stroke">#FFFFFF</CssParameter>
9           <CssParameter name="stroke-width">2</CssParameter>
10        </Stroke>
11      </PolygonSymbolizer>
12      <TextSymbolizer>
```

```
        <Label>
          <ogc:PropertyName>name</ogc:PropertyName>
        </Label>
        <Halo>
          <Radius>3</Radius>
          <Fill>
            <CssParameter name="fill">#FFFFFF</CssParameter>
          </Fill>
        </Halo>
      </TextSymbolizer>
    </Rule>
  </FeatureTypeStyle>
```

Details

This example is similar to the *Polygon with default label*, with the addition of a halo around the labels on **lines 16-21**. A halo creates a color buffer around the label to improve label legibility. **Line 17** sets the radius of the halo, extending the halo 3 pixels around the edge of the label, and **line 19** sets the color of the halo to white (`#FFFFFF`). Since halos are most useful when set to a sharp contrast relative to the text color, this example uses a white halo around black text to ensure optimum readability.

Polygon with styled label

This example improves the label style from the *Polygon with default label* example by centering the label on the polygon, specifying a different font name and size, and setting additional label placement optimizations.

Note: The label placement optimizations discussed below (the `<VendorOption>` tags) are SLD extensions that are custom to GeoServer. They are not part of the SLD 1.0 specification.

Code

`View and download the full "Polygon with styled label" SLD`

```
  <FeatureTypeStyle>
    <Rule>
      <PolygonSymbolizer>
        <Fill>
          <CssParameter name="fill">#40FF40</CssParameter>
        </Fill>
        <Stroke>
          <CssParameter name="stroke">#FFFFFF</CssParameter>
          <CssParameter name="stroke-width">2</CssParameter>
        </Stroke>
      </PolygonSymbolizer>
      <TextSymbolizer>
        <Label>
          <ogc:PropertyName>name</ogc:PropertyName>
        </Label>
        <Font>
          <CssParameter name="font-family">Arial</CssParameter>
```

Fig. 6.57: *Polygon with styled label*

```
18            <CssParameter name="font-size">11</CssParameter>
19            <CssParameter name="font-style">normal</CssParameter>
20            <CssParameter name="font-weight">bold</CssParameter>
21          </Font>
22          <LabelPlacement>
23            <PointPlacement>
24              <AnchorPoint>
25                <AnchorPointX>0.5</AnchorPointX>
26                <AnchorPointY>0.5</AnchorPointY>
27              </AnchorPoint>
28            </PointPlacement>
29          </LabelPlacement>
30          <Fill>
31            <CssParameter name="fill">#000000</CssParameter>
32          </Fill>
33          <VendorOption name="autoWrap">60</VendorOption>
34          <VendorOption name="maxDisplacement">150</VendorOption>
35        </TextSymbolizer>
36      </Rule>
37    </FeatureTypeStyle>
```

Details

This example is similar to the *Polygon with default label* example, with additional styling options within the `<TextSymbolizer>` on lines **12-35**. **Lines 16-21** set the font styling. **Line 17** sets the font family to be Arial, **line 18** sets the font size to 11 pixels, **line 19** sets the font style to "normal" (as opposed to "italic" or "oblique"), and **line 20** sets the font weight to "bold" (as opposed to "normal").

The `<LabelPlacement>` tag on **lines 22-29** affects where the label is placed relative to the centroid of the polygon. **Line 21** centers the label by positioning it 50% (or 0.5) of the way horizontally along the centroid of the polygon. **Line 22** centers the label vertically in exactly the same way.

Finally, there are two added touches for label placement optimization: **line 33** ensures that long labels are split across multiple lines by setting line wrapping on the labels to 60 pixels, and **line 34** allows the label to be displaced by up to 150 pixels. This ensures that labels are compacted and less likely to spill over polygon boundaries. Notice little Massive County in the corner, whose label is now displayed."

Attribute-based polygon

This example styles the polygons differently based on the "pop" (Population) attribute.

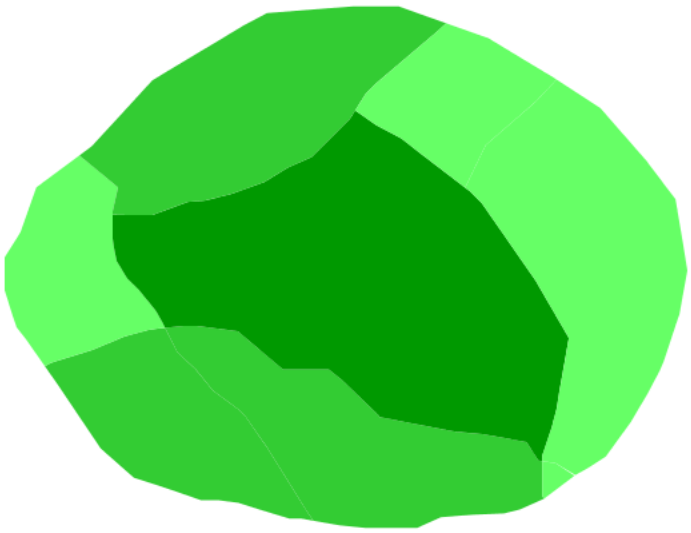


Fig. 6.58: *Attribute-based polygon*

Code

View and download the full "Attribute-based polygon" SLD

```
1    <FeatureTypeStyle>
2      <Rule>
3        <Name>SmallPop</Name>
4        <Title>Less Than 200,000</Title>
5        <ogc:Filter>
6          <ogc:PropertyIsLessThan>
7            <ogc:PropertyName>pop</ogc:PropertyName>
8            <ogc:Literal>200000</ogc:Literal>
9          </ogc:PropertyIsLessThan>
10       </ogc:Filter>
11       <PolygonSymbolizer>
12         <Fill>
13           <CssParameter name="fill">#66FF66</CssParameter>
14         </Fill>
15       </PolygonSymbolizer>
16     </Rule>
17     <Rule>
18       <Name>MediumPop</Name>
```

```
      <Title>200,000 to 500,000</Title>
      <ogc:Filter>
        <ogc:And>
          <ogc:PropertyIsGreaterThanOrEqualTo>
            <ogc:PropertyName>pop</ogc:PropertyName>
            <ogc:Literal>200000</ogc:Literal>
          </ogc:PropertyIsGreaterThanOrEqualTo>
          <ogc:PropertyIsLessThan>
            <ogc:PropertyName>pop</ogc:PropertyName>
            <ogc:Literal>500000</ogc:Literal>
          </ogc:PropertyIsLessThan>
        </ogc:And>
      </ogc:Filter>
      <PolygonSymbolizer>
        <Fill>
          <CssParameter name="fill">#33CC33</CssParameter>
        </Fill>
      </PolygonSymbolizer>
    </Rule>
    <Rule>
      <Name>LargePop</Name>
      <Title>Greater Than 500,000</Title>
      <ogc:Filter>
        <ogc:PropertyIsGreaterThan>
          <ogc:PropertyName>pop</ogc:PropertyName>
          <ogc:Literal>500000</ogc:Literal>
        </ogc:PropertyIsGreaterThan>
      </ogc:Filter>
      <PolygonSymbolizer>
        <Fill>
          <CssParameter name="fill">#009900</CssParameter>
        </Fill>
      </PolygonSymbolizer>
    </Rule>
  </FeatureTypeStyle>
```

Details

Note: Refer to the *Example polygons layer* to see the attributes for the layer. This example has eschewed labels in order to simplify the style, but you can refer to the example *Polygon with styled label* to see which attributes correspond to which polygons.

Each polygon in our fictional country has a population that is represented by the population ("pop") attribute. This style contains three rules that alter the fill based on the value of "pop" attribute, with smaller values yielding a lighter color and larger values yielding a darker color.

The three rules are designed as follows:

Rule order	**Rule name**	**Population** ("pop")	**Color**
1	SmallPop	Less than 200,000	`#66FF66`
2	MediumPop	200,000 to 500,000	`#33CC33`
3	LargePop	Greater than 500,000	`#009900`

The order of the rules does not matter in this case, since each shape is only rendered by a single rule.

The first rule, on **lines 2-16**, specifies the styling of polygons whose population attribute is less than 200,000. **Lines 5-10** set this filter, with **lines 6-9** setting the "less than" filter, **line 7** denoting the attribute ("pop"), and **line 8** the value of 200,000. The color of the polygon fill is set to a light green (`#66FF66`) on **line 13**.

The second rule, on **lines 17-37**, is similar, specifying a style for polygons whose population attribute is greater than or equal to 200,000 but less than 500,000. The filter is set on **lines 20-31**. This filter is longer than in the first rule because two criteria need to be specified instead of one: a "greater than or equal to" and a "less than" filter. Notice the `And` on **line 21** and **line 30**. This mandates that both filters need to be true for the rule to be applicable. The color of the polygon fill is set to a medium green on (`#33CC33`) on **line 34**.

The third rule, on **lines 38-52**, specifies a style for polygons whose population attribute is greater than or equal to 500,000. The filter is set on **lines 41-46**. The color of the polygon fill is the only other difference in this rule, which is set to a dark green (`#009900`) on **line 49**.

Zoom-based polygon

This example alters the style of the polygon at different zoom levels.

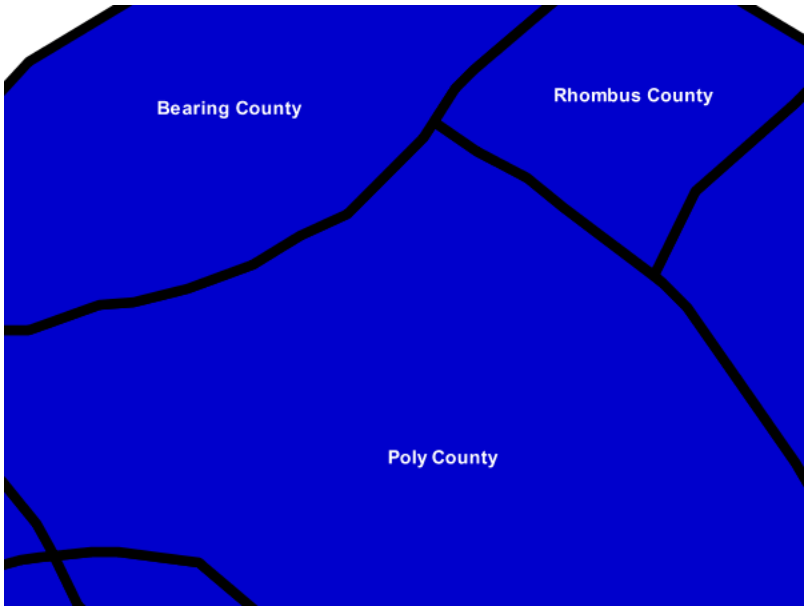


Fig. 6.59: *Zoom-based polygon: Zoomed in*

Code

View and download the full "Zoom-based polygon" SLD

```
1   <FeatureTypeStyle>
2     <Rule>
3       <Name>Large</Name>
4       <MaxScaleDenominator>100000000</MaxScaleDenominator>
5       <PolygonSymbolizer>
6         <Fill>
```

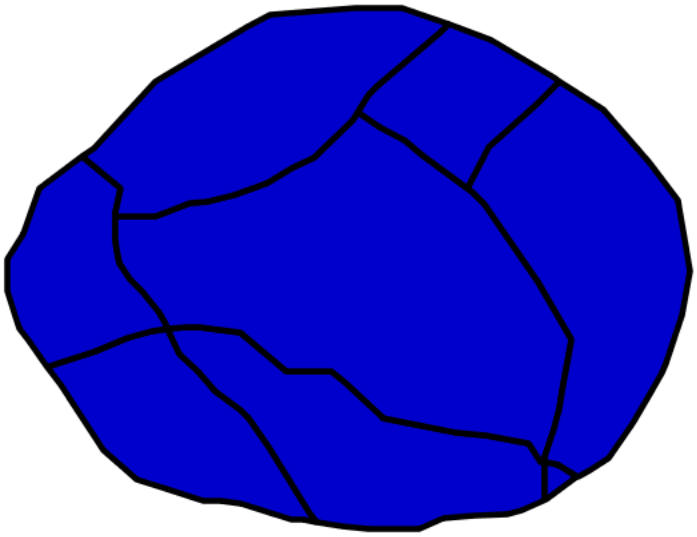


Fig. 6.60: *Zoom-based polygon: Partially zoomed*

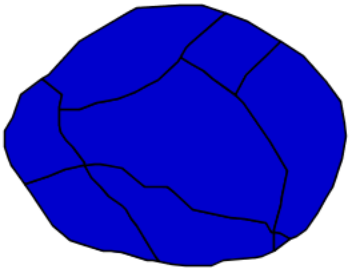


Fig. 6.61: *Zoom-based polygon: Zoomed out*

```
          <CssParameter name="fill">#0000CC</CssParameter>
        </Fill>
        <Stroke>
          <CssParameter name="stroke">#000000</CssParameter>
          <CssParameter name="stroke-width">7</CssParameter>
        </Stroke>
      </PolygonSymbolizer>
      <TextSymbolizer>
        <Label>
          <ogc:PropertyName>name</ogc:PropertyName>
        </Label>
        <Font>
          <CssParameter name="font-family">Arial</CssParameter>
          <CssParameter name="font-size">14</CssParameter>
          <CssParameter name="font-style">normal</CssParameter>
          <CssParameter name="font-weight">bold</CssParameter>
        </Font>
        <LabelPlacement>
          <PointPlacement>
            <AnchorPoint>
              <AnchorPointX>0.5</AnchorPointX>
              <AnchorPointY>0.5</AnchorPointY>
            </AnchorPoint>
          </PointPlacement>
        </LabelPlacement>
        <Fill>
          <CssParameter name="fill">#FFFFFF</CssParameter>
        </Fill>
      </TextSymbolizer>
    </Rule>
    <Rule>
      <Name>Medium</Name>
      <MinScaleDenominator>100000000</MinScaleDenominator>
      <MaxScaleDenominator>200000000</MaxScaleDenominator>
      <PolygonSymbolizer>
        <Fill>
          <CssParameter name="fill">#0000CC</CssParameter>
        </Fill>
        <Stroke>
          <CssParameter name="stroke">#000000</CssParameter>
          <CssParameter name="stroke-width">4</CssParameter>
        </Stroke>
      </PolygonSymbolizer>
    </Rule>
    <Rule>
      <Name>Small</Name>
      <MinScaleDenominator>200000000</MinScaleDenominator>
      <PolygonSymbolizer>
        <Fill>
          <CssParameter name="fill">#0000CC</CssParameter>
        </Fill>
        <Stroke>
          <CssParameter name="stroke">#000000</CssParameter>
          <CssParameter name="stroke-width">1</CssParameter>
        </Stroke>
      </PolygonSymbolizer>
    </Rule>
  </FeatureTypeStyle>
```

Details

It is often desirable to make shapes larger at higher zoom levels when creating a natural-looking map. This example varies the thickness of the lines according to the zoom level. Polygons already do this by nature of being two dimensional, but another way to adjust styling of polygons based on zoom level is to adjust the thickness of the stroke (to be larger as the map is zoomed in) or to limit labels to only certain zoom levels. This is ensures that the size and quantity of strokes and labels remains legible and doesn't overshadow the polygons themselves.

Zoom levels (or more accurately, scale denominators) refer to the scale of the map. A scale denominator of 10,000 means the map has a scale of 1:10,000 in the units of the map projection.

Note: Determining the appropriate scale denominators (zoom levels) to use is beyond the scope of this example.

This style contains three rules, defined as follows:

Rule order	Rule name	Scale denominator	Stroke width	Label display?
1	Large	1:100,000,000 or less	7	Yes
2	Medium	1:100,000,000 to 1:200,000,000	4	No
3	Small	Greater than 1:200,000,000	2	No

The first rule, on **lines 2-36**, is for the smallest scale denominator, corresponding to when the view is "zoomed in". The scale rule is set on **line 40** such that the rule will apply only where the scale denominator is 100,000,000 or less. **Line 7** defines the fill as blue (`#0000CC`). Note that the fill is kept constant across all rules regardless of the scale denominator. As in the *Polygon with default label* or *Polygon with styled label* examples, the rule also contains a `<TextSymbolizer>` at **lines 14-35** for drawing a text label on top of the polygon. **Lines 19-22** set the font information to be Arial, 14 pixels, and bold with no italics. The label is centered both horizontally and vertically along the centroid of the polygon on by setting `<AnchorPointX>` and `<AnchorPointY>` to both be 0.5 (or 50%) on **lines 27-28**. Finally, the color of the font is set to white (`#FFFFFF`) in **line 33**.

The second rule, on **lines 37-50**, is for the intermediate scale denominators, corresponding to when the view is "partially zoomed". The scale rules on **lines 39-40** set the rule such that it will apply to any map with a scale denominator between 100,000,000 and 200,000,000. (The `<MinScaleDenominator>` is inclusive and the `<MaxScaleDenominator>` is exclusive, so a zoom level of exactly 200,000,000 would *not* apply here.) Aside from the scale, there are two differences between this rule and the first: the width of the stroke is set to 4 pixels on **line 47** and a `<TextSymbolizer>` is not present so that no labels will be displayed.

The third rule, on **lines 51-63**, is for the largest scale denominator, corresponding to when the map is "zoomed out". The scale rule is set on **line 53** such that the rule will apply to any map with a scale denominator of 200,000,000 or greater. Again, the only differences between this rule and the others are the width of the lines, which is set to 1 pixel on **line 60**, and the absence of a `<TextSymbolizer>` so that no labels will be displayed.

The resulting style produces a polygon stroke that gets larger as one zooms in and labels that only display when zoomed in to a sufficient level.

Rasters

Rasters are geographic data displayed in a grid. They are similar to image files such as PNG files, except that instead of each point containing visual information, each point contains geographic information in

numerical form. Rasters can be thought of as a georeferenced table of numerical values.

One example of a raster is a Digital Elevation Model (DEM) layer, which has elevation data encoded numerically at each georeferenced data point.

Warning: The code examples shown on this page are **not the full SLD code**, as they omit the SLD header and footer information for the sake of brevity. Please use the links to download the full SLD for each example.

Example raster

The `raster layer` that is used in the examples below contains elevation data for a fictional world. The data is stored in EPSG:4326 (longitude/latitude) and has a data range from 70 to 256. If rendered in grayscale, where minimum values are colored black and maximum values are colored white, the raster would look like this:

Fig. 6.62: *Raster file as rendered in grayscale*

`Download the raster shapefile`

Two-color gradient

This example shows a two-color style with green at lower elevations and brown at higher elevations.

Fig. 6.63: *Two-color gradient*

Code

View and download the full "Two-color gradient" SLD

```
1   <FeatureTypeStyle>
2     <Rule>
3       <RasterSymbolizer>
4         <ColorMap>
5           <ColorMapEntry color="#008000" quantity="70" />
6           <ColorMapEntry color="#663333" quantity="256" />
7         </ColorMap>
8       </RasterSymbolizer>
9     </Rule>
10  </FeatureTypeStyle>
```

Details

There is one `<Rule>` in one `<FeatureTypeStyle>` for this example, which is the simplest possible situation. All subsequent examples will share this characteristic. Styling of rasters is done via the `<RasterSymbolizer>` tag (**lines 3-8**).

This example creates a smooth gradient between two colors corresponding to two elevation values. The gradient is created via the `<ColorMap>` on **lines 4-7**. Each entry in the `<ColorMap>` represents one entry or anchor in the gradient. **Line 5** sets the lower value of 70 via the `quantity` parameter, which is styled a dark green (`#008000`). **Line 6** sets the upper value of 256 via the `quantity` parameter again, which is styled a dark brown (`#663333`). All data values in between these two quantities will be linearly interpolated: a value of 163 (the midpoint between 70 and 256) will be colored as the midpoint between the two colors (in this case approximately `#335717`, a muddy green).

Transparent gradient

This example creates the same two-color gradient as in the *Two-color gradient* as in the example above but makes the entire layer mostly transparent by setting a 30% opacity.

Fig. 6.64: *Transparent gradient*

Code

View and download the full "Transparent gradient" SLD

```
<FeatureTypeStyle>
  <Rule>
    <RasterSymbolizer>
      <Opacity>0.3</Opacity>
      <ColorMap>
        <ColorMapEntry color="#008000" quantity="70" />
        <ColorMapEntry color="#663333" quantity="256" />
      </ColorMap>
    </RasterSymbolizer>
  </Rule>
</FeatureTypeStyle>
```

Details

This example is similar to the *Two-color gradient* example save for the addition of **line 4**, which sets the opacity of the layer to 0.3 (or 30% opaque). An opacity value of 1 means that the shape is drawn 100% opaque, while an opacity value of 0 means that the shape is rendered as completely transparent. The value of 0.3 means that the the raster partially takes on the color and style of whatever is drawn beneath it. Since the background is white in this example, the colors generated from the `<ColorMap>` look lighter, but were the raster imposed on a dark background the resulting colors would be darker.

Brightness and contrast

This example normalizes the color output and then increases the brightness by a factor of 2.

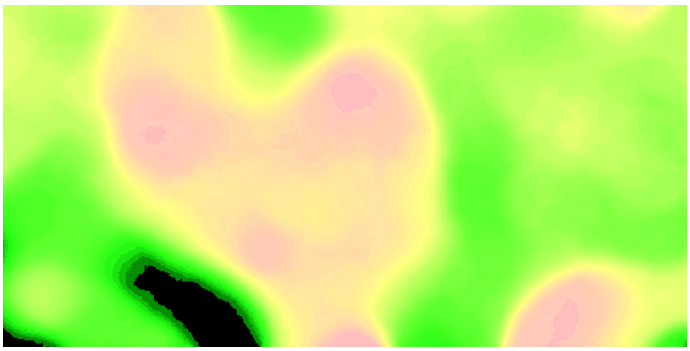


Fig. 6.65: *Brightness and contrast*

Code

`View and download the full "Brightness and contrast" SLD`

```
<FeatureTypeStyle>
  <Rule>
    <RasterSymbolizer>
      <ContrastEnhancement>
        <Normalize />
        <GammaValue>0.5</GammaValue>
      </ContrastEnhancement>
      <ColorMap>
```

```
9             <ColorMapEntry color="#008000" quantity="70" />
10            <ColorMapEntry color="#663333" quantity="256" />
11          </ColorMap>
12        </RasterSymbolizer>
13      </Rule>
14    </FeatureTypeStyle>
```

Details

This example is similar to the *Two-color gradient*, save for the addition of the `<ContrastEnhancement>` tag on **lines 4-7**. **Line 5** normalizes the output by increasing the contrast to its maximum extent. **Line 6** then adjusts the brightness by a factor of 0.5. Since values less than 1 make the output brighter, a value of 0.5 makes the output twice as bright.

As with previous examples, **lines 8-11** determine the `<ColorMap>`, with **line 9** setting the lower bound (70) to be colored dark green (`#008000`) and **line 10** setting the upper bound (256) to be colored dark brown (`#663333`).

Three-color gradient

This example creates a three-color gradient in primary colors.

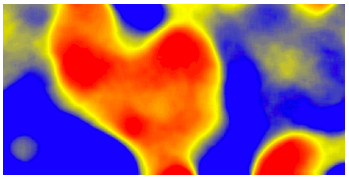


Fig. 6.66: *Three-color gradient*

Code

`View and download the full "Three-color gradient" SLD`

```
1     <FeatureTypeStyle>
2       <Rule>
3         <RasterSymbolizer>
4           <ColorMap>
5             <ColorMapEntry color="#0000FF" quantity="150" />
6             <ColorMapEntry color="#FFFF00" quantity="200" />
7             <ColorMapEntry color="#FF0000" quantity="250" />
8           </ColorMap>
9         </RasterSymbolizer>
10      </Rule>
11    </FeatureTypeStyle>
```

Details

This example creates a three-color gradient based on a `<ColorMap>` with three entries on **lines 4-8**: **line 5** specifies the lower bound (150) be styled in blue (`#0000FF`), **line 6** specifies an intermediate point (200) be

styled in yellow (`#FFFF00`), and **line 7** specifies the upper bound (250) be styled in red (`#FF0000`).

Since our data values run between 70 and 256, some data points are not accounted for in this style. Those values below the lowest entry in the color map (the range from 70 to 149) are styled the same color as the lower bound, in this case blue. Values above the upper bound in the color map (the range from 251 to 256) are styled the same color as the upper bound, in this case red.

Alpha channel

This example creates an "alpha channel" effect such that higher values are increasingly transparent.

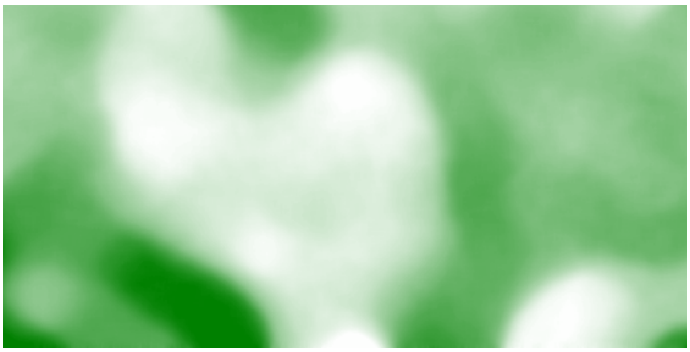


Fig. 6.67: *Alpha channel*

Code

View and download the full "Alpha channel" SLD

```
1   <FeatureTypeStyle>
2     <Rule>
3       <RasterSymbolizer>
4         <ColorMap>
5           <ColorMapEntry color="#008000" quantity="70" />
6           <ColorMapEntry color="#008000" quantity="256" opacity="0"/>
7         </ColorMap>
8       </RasterSymbolizer>
9     </Rule>
10  </FeatureTypeStyle>
```

Details

An alpha channel is another way of referring to variable transparency. Much like how a gradient maps values to colors, each entry in a `<ColorMap>` can have a value for opacity (with the default being 1.0 or completely opaque).

In this example, there is a `<ColorMap>` with two entries: **line 5** specifies the lower bound of 70 be colored dark green (`#008000`), while **line 6** specifies the upper bound of 256 also be colored dark green but with an opacity value of 0. This means that values of 256 will be rendered at 0% opacity (entirely transparent). Just like the gradient color, the opacity is also linearly interpolated such that a value of 163 (the midpoint between 70 and 256) is rendered at 50% opacity.

Discrete colors

This example shows a gradient that is not linearly interpolated but instead has values mapped precisely to one of three specific colors.

Note: This example leverages an SLD extension in GeoServer. Discrete colors are not part of the standard SLD 1.0 specification.

Fig. 6.68: *Discrete colors*

Code

View and download the full "Discrete colors" SLD

```
1   <FeatureTypeStyle>
2     <Rule>
3       <RasterSymbolizer>
4         <ColorMap type="intervals">
5           <ColorMapEntry color="#008000" quantity="150" />
6           <ColorMapEntry color="#663333" quantity="256" />
7         </ColorMap>
8       </RasterSymbolizer>
9     </Rule>
10  </FeatureTypeStyle>
```

Details

Sometimes color bands in discrete steps are more appropriate than a color gradient. The `type="intervals"` parameter added to the `<ColorMap>` on **line 4** sets the display to output discrete colors instead of a gradient. The values in each entry correspond to the upper bound for the color band such that colors are mapped to values less than the value of one entry but greater than or equal to the next lower entry. For example, **line 5** colors all values less than 150 to dark green (`#008000`) and line 6 colors all values less than 256 but greater than or equal to 150 to dark brown (`#663333`).

Many color gradient

This example shows a gradient interpolated across eight different colors.

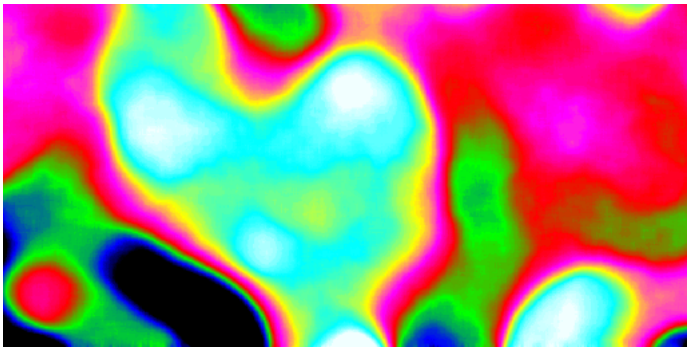


Fig. 6.69: *Many color gradient*

Code

View and download the full "Many color gradient" SLD

```
1   <FeatureTypeStyle>
2     <Rule>
3       <RasterSymbolizer>
4         <ColorMap>
5           <ColorMapEntry color="#000000" quantity="95" />
6           <ColorMapEntry color="#0000FF" quantity="110" />
7           <ColorMapEntry color="#00FF00" quantity="135" />
8           <ColorMapEntry color="#FF0000" quantity="160" />
9           <ColorMapEntry color="#FF00FF" quantity="185" />
10          <ColorMapEntry color="#FFFF00" quantity="210" />
11          <ColorMapEntry color="#00FFFF" quantity="235" />
12          <ColorMapEntry color="#FFFFFF" quantity="256" />
13        </ColorMap>
14      </RasterSymbolizer>
15    </Rule>
16  </FeatureTypeStyle>
```

Details

A `<ColorMap>` can include up to 255 `<ColorMapEntry>` elements. This example has eight entries (**lines 4-13**):

Entry number	Value	Color	RGB code
1	95	Black	`#000000`
2	110	Blue	`#0000FF`
3	135	Green	`#00FF00`
4	160	Red	`#FF0000`
5	185	Purple	`#FF00FF`
6	210	Yellow	`#FFFF00`
7	235	Cyan	`#00FFFF`
8	256	White	`#FFFFFF`

6.2.4 SLD Reference

The OGC **Styled Layer Descriptor (SLD)** standard defines a language for expressing styling of geospatial data. GeoServer uses SLD as its primary styling language.

SLD 1.0.0 is defined in the following specification:

- OGC Styled Layer Descriptor Implementation Specification, Version 1.0.0

Subsequently the functionality of SLD has been split into two specifications:

- OGC Symbology Encoding Implementation Specification, Version 1.1.0
- OGC Styled Layer Descriptor profile of the Web Map Service Implementation Specification, Version 1.1.0

GeoServer implements the SLD 1.0.0 standard, as well as some parts of the SE 1.1.0 and WMS-SLD 1.1.0 standards.

Elements of SLD

The following sections describe the SLD elements implemented in GeoServer.

The root element for an SLD is `<StyledLayerDescriptor>`. It contains a **Layers** and **Styles** elements which describe how a map is to be composed and styled.

StyledLayerDescriptor

The root element for an SLD is `<StyledLayerDescriptor>`. It contains a sequence of *Layers* defining the styled map content.

The `<StyledLayerDescriptor>` element contains the following elements:

Tag	Required?	Description
`<NamedLayer>`	0..N	A reference to a named layer in the server catalog
`<UserLayer>`	0..N	A layer defined in the style itself

Layers

An SLD document contains a sequence of layer definitions indicating the layers to be styled. Each layer definition is either a **NamedLayer** reference or a supplied **UserLayer**.

NamedLayer

A **NamedLayer** specifies an existing layer to be styled, and the styling to apply to it. The styling may be any combination of catalog styles and explicitly-defined styles. If no style is specified, the default style for the layer is used.

The `<NamedLayer>` element contains the following elements:

Tag	Required?	Description
`<Name>`	Yes	The name of the layer to be styled. (Ignored in catalog styles.)
`<Description>`	No	The description for the layer.
`<NamedStyle>`	0..N	The name of a catalog style to apply to the layer.
`<UserStyle>`	0..N	The definition of a style to apply to the layer. See *Styles*

UserLayer

A **UserLayer** defines a new layer to be styled, and the styling to apply to it. The data for the layer is provided directly in the layer definition using the `<InlineFeature>` element. Since the layer is not known to the server, the styling must be explicitly specified as well.

The `<UserLayer>` element contains the following elements:

Tag	Required?	Description
`<Name>`	No	The name for the layer being defined
`<Description>`	No	The description for the layer
`<InlineFeature>`	No	One or more feature collections providing the layer data, specified using GML.
`<UserStyle>`	1..N	The definition of the style(s) to use for the layer. See *Styles*

A common use is to define a geometry to be rendered to indicate an Area Of Interest.

InlineFeature

An **InlineFeature** element contains data defining a layer to be styled. The element contains one or more `<FeatureCollection>` elements defining the data. Each Feature Collection can contain any number of `<featureMember>` elements, each containing a feature specified using GML markup. The features can contain any type of geometry (point, line or polygon, and collections of these). They may also contain scalar-valued attributes, which can be useful for labelling.

Example

The following style specifies a named layer using the default style, and a user-defined layer with inline data and styling. It displays the US States layer, with a labelled red box surrounding the Pacific NW.

```
<sld:StyledLayerDescriptor xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.opengis.net/sld StyledLayerDescriptor.xsd"
   xmlns:gml="http://www.opengis.net/gml/3.2" xmlns:ogc="http://www.opengis.net/ogc"
   xmlns:sld="http://www.opengis.net/sld" version="1.0.0">
   <sld:NamedLayer>
      <sld:Name>usa:states</sld:Name>
   </sld:NamedLayer>
   <sld:UserLayer>
      <sld:Name>Inline</sld:Name>
      <sld:InlineFeature>
         <sld:FeatureCollection>
            <sld:featureMember>
              <feature>
                <geometryProperty>
                  <gml:Polygon>
                     <gml:outerBoundaryIs>
                        <gml:LinearRing>
                           <gml:coordinates>
          -127.0,51.0 -110.0,51.0 -110.0,41.0 -127.0,41.0 -127.0,51.0
                           </gml:coordinates>
                        </gml:LinearRing>
                     </gml:outerBoundaryIs>
                  </gml:Polygon>
```

```
                </geometryProperty>
                <title>Pacific NW </title>
              </feature>
            </sld:featureMember>
         </sld:FeatureCollection>
      </sld:InlineFeature>
      <sld:UserStyle>
         <sld:FeatureTypeStyle>
            <sld:Rule>
                  <sld:PolygonSymbolizer>
                <Stroke>
                  <CssParameter name="stroke">#FF0000</CssParameter>
                  <CssParameter name="stroke-width">2</CssParameter>
                </Stroke>
              </sld:PolygonSymbolizer>
              <sld:TextSymbolizer>
                <sld:Label>
                  <ogc:PropertyName>title</ogc:PropertyName>
                </sld:Label>
                <sld:Fill>
                  <sld:CssParameter name="fill">#FF0000</sld:CssParameter>
                </sld:Fill>
              </sld:TextSymbolizer>
            </sld:Rule>
         </sld:FeatureTypeStyle>
      </sld:UserStyle>
   </sld:UserLayer>
</sld:StyledLayerDescriptor>
```

Styles

The style elements specify the styling to be applied to a layer.

UserStyle

The **UserStyle** element defines styling for a layer.

The `<UserStyle>` element contains the following elements:

Tag	Required?	Description
`<Name>`	No	The name of the style, used to reference it externally. (Ignored for catalog styles.)
`<Title>`	No	The title of the style.
`<Abstract>`	No	The description for the style.
`<IsDefault>`	No	Whether the style is the default one for a named layer. Used in SLD **Library Mode**. Values are `1` or `0` (default).
`<FeatureTypeStyle>`	1..N	Defines the symbology for rendering a single feature type.

FeatureTypeStyle

The **FeatureTypeStyle** element specifies the styling that is applied to a single feature type of a layer. It contains a list of rules which determine the symbology to be applied to each feature of a layer.

The <FeatureTypeStyle> element contains the following elements:

Tag	Required?	Description
<Name>	No	Not used at present
<Title>	No	The title for the style.
<Abstract>	No	The description for the style.
<FeatureTypeName>	No	Identifies the feature type the style is to be applied to. Omitted if the style applies to all features in a layer.
<Rule>	1..N	A styling rule to be evaluated. See *Rules*

Usually a layer contains only a single feature type, so the <FeatureTypeName> is omitted.

Any number of <FeatureTypeStyle> elements can be specified in a style. In GeoServer each one is rendered into a separate image buffer. After all features are rendered the buffers are composited to form the final layer image. The compositing is done in the order the FeatureTypeStyles are given in the SLD, with the first one on the bottom (the "Painter's Model"). This effectively creates "virtual layers", which can be used to achieve styling effects such as cased lines.

Styles contain **Rules** and **Filters** to determine sets of features to be styled with specific symbology. Rules may also specify the scale range in which the feature styling is visible.

Rules

Styling **rules** define the portrayal of features. A rule combines a *filter* with any number of symbolizers. Features for which the filter condition evaluates as true are rendered using the the symbolizers in the rule.

Syntax

The <Rule> element contains the following elements:

Tag	Required?	Description
<Name>	No	Specifies a name for the rule.
<Title>	No	Specifies a title for the rule. The title is used in display lists and legends.
<Abstract>	No	Specifies an abstract describing the rule.
<Filter>	No	Specifies a filter controlling when the rule is applied. See *Filters*
<MinScaleDenominator>	No	Specifies the minimum scale denominator (inclusive) for the scale range in which this rule applies. If present, the rule applies at the given scale and all smaller scales.
<MaxScaleDenominator>	No	Specifies the maximum scale denominator (exclusive) for the scale range in which this rule applies. If present, the rule applies at scales larger than the given scale.
<PointSymbolizer>	0..N	Specifies styling as points. See *PointSymbolizer*
<LineSymbolizer>	0..N	Specifies styling as lines. See *LineSymbolizer*
<PolygonSymbolizer>	0..N	Specifies styling as polygons. See *PolygonSymbolizer*
<TextSymbolizer>	0..N	Specifies styling for text labels. See *TextSymbolizer*
<RasterSymbolizer>	0..N	Specifies styling for raster data. See *RasterSymbolizer*

Scale Selection

Rules support **scale selection** to allow specifying the scale range in which a rule may be applied (assuming the filter condition is satisfied as well, if present). Scale selection allows for varying portrayal of features at different map scales. In particular, at smaller scales it is common to use simpler styling for features, or even prevent the display of some features altogether.

Scale ranges are specified by using **scale denominators**. These values correspond directly to the ground distance covered by a map, but are inversely related to the common "large" and "small" terminology for map scale. In other words:

- **large scale** maps cover *less* area and have a *smaller* scale denominator
- **small scale** maps cover *more* area and have a *larger* scale denominator

Two optional elements specify the scale range for a rule:

Tag	Required?	Description
`<MinScaleDenominator>`	No	Specifies the minimum scale denominator (inclusive) for the scale range in which this rule applies. If present, the rule applies at the given scale and all smaller scales.
`<MaxScaleDenominator>`	No	Specifies the maximum scale denominator (exclusive) for the scale range in which this rule applies. If present, the rule applies at scales larger than the given scale.

Note: The current scale can also be obtained via the `wms_scale_denominator` *SLD environment variable*. This allows including scale dependency in *Filter Expressions*.

The following example shows the use of scale selection in a pair of rules. The rules specify that:

- at scales **above** 1:20,000 (*larger* scales, with scale denominators *smaller* than 20,000) features are symbolized with 10-pixel red squares,
- at scales **at or below** 1:20,000 (*smaller* scales, with scale denominators *larger* than 20,000) features are symbolized with 4-pixel blue triangles.

```
<Rule>
   <MaxScaleDenominator>20000</MaxScaleDenominator>
   <PointSymbolizer>
     <Graphic>
       <Mark>
         <WellKnownName>square</WellKnownName>
         <Fill><CssParameter name="fill">#FF0000</CssParameter>
       </Mark>
       <Size>10</Size>
     </Graphic>
   </PointSymbolizer>
</Rule>
<Rule>
   <MinScaleDenominator>20000</MinScaleDenominator>
   <PointSymbolizer>
     <Graphic>
       <Mark>
         <WellKnownName>triangle</WellKnownName>
         <Fill><CssParameter name="fill">#0000FF</CssParameter>
       </Mark>
```

```
      <Size>4</Size>
    </Graphic>
  </PointSymbolizer>
</Rule>
```

Evaluation Order

Within an SLD document each `<FeatureTypeStyle>` can contain many rules. Multiple-rule SLDs are the basis for thematic styling. In GeoServer each `<FeatureTypeStyle>` is evaluated once for each feature processed. The rules within it are evaluated in the order they occur. A rule is applied when its filter condition (if any) is true for a feature and the rule is enabled at the current map scale. The rule is applied by rendering the feature using each symbolizer within the rule, in the order in which they occur. The rendering is performed into the image buffer for the parent `<FeatureTypeStyle>`. Thus symbolizers earlier in a `FeatureTypeStyle` and `Rule` are rendered *before* symbolizers occuring later in the document (this is the "Painter's Model" method of rendering).

Examples

The following rule applies only to features which have a `POPULATION` attribute greater than 100,000, and symbolizes the features as red points.

```
<Rule>
  <ogc:Filter>
    <ogc:PropertyIsGreaterThan>
      <ogc:PropertyName>POPULATION</ogc:PropertyName>
      <ogc:Literal>100000</ogc:Literal>
    </ogc:PropertyIsGreaterThan>
  </ogc:Filter>
  <PointSymbolizer>
    <Graphic>
      <Mark>
        <Fill><CssParameter name="fill">#FF0000</CssParameter>
      </Mark>
    </Graphic>
  </PointSymbolizer>
</Rule>
```

An additional rule can be added which applies to features whose `POPULATION` attribute is less than 100,000, and symbolizes them as green points.

```
<Rule>
  <ogc:Filter>
    <ogc:PropertyIsLessThan>
      <ogc:PropertyName>POPULATION</ogc:PropertyName>
      <ogc:Literal>100000</ogc:Literal>
    </ogc:PropertyIsLessThan>
  </ogc:Filter>
  <PointSymbolizer>
    <Graphic>
      <Mark>
        <Fill><CssParameter name="fill">#0000FF</CssParameter>
      </Mark>
    </Graphic>
```

```
  </PointSymbolizer>
</Rule>
```

Filters

A *filter* is the mechanism in SLD for specifying conditions. They are similar in functionality to the SQL "WHERE" clause. Filters are used within *Rules* to determine which styles should be applied to which features in a data set. The filter language used by SLD follows the OGC Filter Encoding standard. It is described in detail in the *Filter Encoding Reference*.

A filter condition is specified by using a **comparison operator** or a **spatial operator**, or two or more of these combined by **logical operators**. The operators are usually used to compare properties of the features being filtered to other properties or to literal data.

Comparison operators

Comparison operators are used to specify conditions on the non-spatial attributes of a feature. The following **binary comparison operators** are available:

- `<PropertyIsEqualTo>`
- `<PropertyIsNotEqualTo>`
- `<PropertyIsLessThan>`
- `<PropertyIsLessThanOrEqualTo>`
- `<PropertyIsGreaterThan>`
- `<PropertyIsGreaterThanOrEqualTo>`

These operators contain two *filter expressions* to be compared. The first operand is often a `<PropertyName>`, but both operands may be any expression, function or literal value.

Binary comparison operators may include a `matchCase` attribute with the value `true` or `false`. If this attribute is `true` (which is the default), string comparisons are case-sensitive. If the attribute is specified and has the value `false` strings comparisons do not check case.

Other available **value comparison operators** are:

- `<PropertyIsLike>`
- `<PropertyIsNull>`
- `<PropertyIsBetween>`

`<PropertyIsLike>` matches a string property value against a text **pattern**. It contains a `<PropertyName>` element containing the name of the property containing the string to be matched and a `<Literal>` element containing the pattern. The pattern is specified by a sequence of regular characters and three special pattern characters. The pattern characters are defined by the following required attributes of the `<PropertyIsLike>` element:

- `wildCard` specifies a pattern character which matches any sequence of zero or more characters
- `singleChar` specifies a pattern character which matches any single character
- `escapeChar` specifies an escape character which can be used to escape these pattern characters

<PropertyIsNull> tests whether a property value is null. It contains a single <PropertyName> element containing the name of the property containing the value to be tested.

<PropertyIsBetween> tests whether an expression value lies within a range. It contains a *filter expression* providing the value to test, followed by the elements <LowerBoundary> and <UpperBoundary>, each containing a *filter expression*.

Examples

- The following filter selects features whose NAME attribute has the value of "New York":

```
<PropertyIsEqualTo>
   <PropertyName>NAME</PropertyName>
   <Literal>New York</Literal>
</PropertyIsEqualTo>
```

- The following filter selects features whose geometry area is greater than 1,000,000:

```
<PropertyIsGreaterThan>
   <ogc:Function name="area">
     <PropertyName>GEOMETRY</PropertyName>
   </ogc:Function>
   <Literal>1000000</Literal>
</PropertyIsEqualTo>
```

Spatial operators

Spatial operators are used to specify conditions on the geometric attributes of a feature. The following spatial operators are available:

Topological Operators

These operators test topological spatial relationships using the standard OGC Simple Features predicates:

- <Intersects>
- <Equals>
- <Disjoint>
- <Touches>
- <Within>
- <Overlaps>
- <Crosses>
- <Intersects>
- <Contains>

The content for these operators is a <PropertyName> element for a geometry-valued property and a GML geometry literal.

Distance Operators

These operators compute distance relationships between geometries:

- <DWithin>

- `<Beyond>`

The content for these elements is a `<PropertyName>` element for a geometry-valued property, a GML geometry literal, and a `<Distance>` element containing the value for the distance tolerance. The `<Distance>` element may include an optional `units` attribute.

Bounding Box Operator

This operator tests whether a feature geometry attribute intersects a given bounding box:

- `<BBOX>`

The content is an optional `<PropertyName>` element, and a GML envelope literal. If the `PropertyName` is omitted the default geometry attribute is assumed.

Examples

- The following filter selects features with a geometry that intersects the point (1,1):

```
<Intersects>
   <PropertyName>GEOMETRY</PropertyName>
   <Literal>
      <gml:Point>
         <gml:coordinates>1 1</gml:coordinates>
      </gml:Point>
   </Literal>
</Intersects>
```

- The following filter selects features with a geometry that intersects the box [-10,0 : 10,10]:

```
<ogc:BBOX>
  <ogc:PropertyName>GEOMETRY</ogc:PropertyName>
  <gml:Box srsName="urn:x-ogc:def:crs:EPSG:4326">
    <gml:coord>
      <gml:X>-10</gml:X> <gml:Y>0</gml:Y>
    </gml:coord>
    <gml:coord>
      <gml:X>10</gml:X> <gml:Y>10</gml:Y>
    </gml:coord>
  </gml:Box>
</ogc:BBOX>
```

Logical operators

Logical operators are used to create logical combinations of other filter operators. They may be nested to any depth. The following logical operators are available:

- `<And>`
- `<Or>`
- `<Not>`

The content for `<And>` and `<Or>` is two filter operator elements. The content for `<Not>` is a single filter operator element.

Examples

- The following filter uses `<And>` to combine a comparison operator and a spatial operator:

```
<And>
   <PropertyIsEqualTo>
      <PropertyName>NAME</PropertyName>
      <Literal>New York</Literal>
   </PropertyIsEqualTo>
   <Intersects>
      <PropertyName>GEOMETRY</PropertyName>
      <Literal>
         <gml:Point>
             <gml:coordinates>1 1</gml:coordinates>
         </gml:Point>
      </Literal>
   </Intersects>
</And>
```

Filter Expressions

Filter expressions allow performing computation on data values. The following elements can be used to form expressions.

Arithmetic Operators

These operators perform arithmetic on numeric values. Each contains two expressions as sub-elements.

- `<Add>`
- `<Sub>`
- `<Mul>`
- `<Div>`

Functions

The `<Function>` element specifies a filter function to be evaluated. The `name` attribute gives the function name. The element contains a sequence of zero or more filter expressions providing the function arguments. See the *Filter Function Reference* for details of the functions provided by GeoServer.

Feature Property Values

The `<PropertyName>` element allows referring to the value of a given feature attribute. It contains a string specifying the attribute name.

Literals

The `<Literal>` element allows specifying constant values of numeric, boolean, string, date or geometry type.

Rules contain **Symbolizers** to specify how features are styled. There are 5 types of symbolizers:

- `PointSymbolizer`, which styles features as **points**
- `LineSymbolizer`, which styles features as **lines**
- `PolygonSymbolizer`, which styles features as **polygons**
- `TextSymbolizer`, which styles **text labels** for features

- `RasterSymbolizer`, which styles **raster coverages**

Each symbolizer type has its own parameters to control styling.

PointSymbolizer

A **PointSymbolizer** styles features as **points**. Points are depicted as graphic symbols at a single location on the map.

Syntax

A `<PointSymbolizer>` contains an optional `<Geometry>` element, and a required `<Graphic>` element specifying the point symbology.

Tag	Required?	Description
`<Geometry>`	No	Specifies the geometry to be rendered.
`<Graphic>`	Yes	Specifies the styling for the point symbol.

Geometry

The `<Geometry>` element is optional. If present, it specifies the featuretype property from which to obtain the geometry to style using a `<PropertyName>` element. See also *Geometry transformations in SLD* for GeoServer extensions for specifying geometry.

Any kind of geometry may be styled with a `<PointSymbolizer>`. For non-point geometries, a representative point is used (such as the centroid of a line or polygon).

Graphic

Symbology is specified using a `<Graphic>` element. The symbol is specified by either an `<ExternalGraphic>` or a `<Mark>` element. **External Graphics** are image files (in a format such as PNG or SVG) that contain the shape and color information defining how to render a symbol. **Marks** are vector shapes whose stroke and fill are defined explicitly in the symbolizer.

There are five possible sub-elements of the `<Graphic>` element. One of `<ExternalGraphic>` or `<Mark>` must be specified; the others are optional.

Tag	Required?	Description
`<ExternalGraphic>`	No (when using `<Mark>`)	Specifies an external image file to use as the symbol.
`<Mark>`	No (when using `<ExternalGraphic>`)	Specifies a named shape to use as the symbol.
`<Opacity>`	No	Specifies the opacity (transparency) of the symbol. Values range from `0` (completely transparent) to `1` (completely opaque). Value may contain *expressions*. Default is `1` (opaque).
`<Size>`	No	Specifies the size of the symbol, in pixels. When used with an image file, this specifies the height of the image, with the width being scaled accordingly. if omitted the native symbol size is used. Value may contain *expressions*.
`<Rotation>`	No	Specifies the rotation of the symbol about its center point, in decimal degrees. Positive values indicate rotation in the clockwise direction, negative values indicate counter-clockwise rotation. Value may contain *expressions*. Default is `0`.

ExternalGraphic

External Graphics are image files (in formats such as PNG or SVG) that contain the shape and color information defining how to render a symbol. For GeoServer extensions for specifying external graphics, see *Graphic symbology in GeoServer*.

The `<ExternalGraphic>` element has the sub-elements:

Tag	Required?	Description
`<OnlineResource>`	Yes	The `xlink:href` attribute specifies the location of the image file. The value can be either a URL or a local pathname relative to the SLD directory. The value can contain CQL expressions delimited by `${ }`. The attribute `xlink:type="simple"` is also required. The element does not contain any content.
`<Format>`	Yes	The MIME type of the image format. Most standard web image formats are supported. Common MIME types are `image/png`, `image/jpeg`, `image/gif`, and `image/svg+xml`

Mark

Marks are predefined vector shapes identified by a well-known name. Their fill and stroke can be defined explicitly in the SLD. For GeoServer extensions for specifying mark symbols, see *Graphic symbology in GeoServer*.

The `<Mark>` element has the sub-elements:

Tag	Required?	Description
<WellKnownName>	No	The name of the shape. Standard SLD shapes are circle, square, triangle, star, cross, or x. Default is square.
<Fill>	No	Specifies how the symbol should be filled (for closed shapes). Options are to use <CssParameter name="fill"> to specify a solid fill color, or using <GraphicFill> for a tiled graphic fill. See the PolygonSymbolizer *Fill* for the full syntax.
<Stroke>	No	Specifies how the symbol linework should be drawn. Some options are using <CssParameter name="stroke"> to specify a stroke color, or using <GraphicStroke> for a repeated graphic. See the LineSymbolizer *Stroke* for the full syntax.

Example

The following symbolizer is taken from the *Points* section in the *SLD Cookbook*.

```
<PointSymbolizer>
  <Graphic>
    <Mark>
      <WellKnownName>circle</WellKnownName>
      <Fill>
        <CssParameter name="fill">#FF0000</CssParameter>
      </Fill>
    </Mark>
    <Size>6</Size>
  </Graphic>
</PointSymbolizer>
```

The symbolizer contains the required <Graphic> element. Inside this element is the <Mark> element and <Size> element, which are the minimum required element inside <Graphic> (when not using the <ExternalGraphic> element). The <Mark> element contains the <WellKnownName> element and a <Fill> element. No other element are required. In summary, this example specifies the following:

1. Features will be rendered as points
2. Points will be rendered as circles
3. Circles will be rendered with a diameter of 6 pixels and filled with the color red

The next example uses an external graphic loaded from the file system:

```
<PointSymbolizer>
  <Graphic>
    <ExternalGraphic>
      <OnlineResource xlink:type="simple"
                      xlink:href="file:///var/www/htdocs/sun.png" />
      <Format>image.png</Format>
    </ExternalGraphic>
  </Graphic>
</PointSymbolizer>
```

For file:// URLs, the file must be readable by the user the GeoServer process is running as. You can also use href:// URLs to reference remote graphics.

Further examples can be found in the *Points* section of the *SLD Cookbook*.

Using expressions in parameter values

Many SLD parameters allow their values to be of **mixed type**. This means that the element content can be:

- a constant value expressed as a string
- a *filter expression*
- any combination of strings and filter expressions.

Using expressions in parameter values provides the ability to determine styling dynamically on a per-feature basis, by computing parameter values from feature properties. Using computed parameters is an alternative to using rules in some situations, and may provide a more compact SLD document.

GeoServer also supports using substitution variables provided in WMS requests. This is described in *Variable substitution in SLD*.

LineSymbolizer

A **LineSymbolizer** styles features as **lines**. Lines are one-dimensional geometries that have both position and length. Each line is comprised of one or more **line segments**, and has either two **ends** or none (if it is closed).

Syntax

A `<LineSymbolizer>` contains an optional `<Geometry>` element, and a required `<Stroke>` element specifying the line symbology.

Tag	Required?	Description
`<Geometry>`	No	Specifies the geometry to be rendered.
`<Stroke>`	Yes	Specifies the styling for the line.
`<PerpendicularOffset>`	No	Specifies the perpendicular offset for the current line

Geometry

The `<Geometry>` element is optional. If present, it specifies the featuretype property from which to obtain the geometry to style using the `PropertyName` element. See also *Geometry transformations in SLD* for GeoServer extensions for specifying geometry.

Any kind of geometry may be styled with a `<LineSymbolizer>`. Point geometries are treated as lines of zero length, with a horizontal orientation. For polygonal geometries the boundary (or boundaries) are used as the lines, each line being a closed ring with no ends.

Stroke

The `<Stroke>` element specifies the styling of a line. There are three elements that can be included inside the `<Stroke>` element.

Tag	Required?	Description
`<GraphicFill>`	No	Renders the pixels of the line with a repeated pattern.
`<GraphicStroke>`	No	Renders the line with a repeated linear graphic.
`<CssParameter>`	0..N	Determines the stroke styling parameters.

GraphicFill

The `<GraphicFill>` element specifies that the pixels of the line are to be filled with a repeating graphic image or symbol. The graphic is specified by a `<Graphic>` sub-element, which is described in the `PointSymbolizer` *Graphic* section.

GraphicStroke

The `<GraphicStroke>` element specifies the the line is to be drawn using a repeated graphic image or symbol following the line. The graphic is specified by a `<Graphic>` sub-element, which is described in the `PointSymbolizer` *Graphic* section.

The spacing of the graphic symbol can be specified using the `<Size>` element in the `<Graphic>` element, or the `<CSSParameter name="stroke-dasharray">` in the `Stroke` element.

CssParameter

The `<CssParameter>` elements describe the basic styling of the line. Any number of `<CssParameter>` elements can be specified.

The `name` **attribute** indicates what aspect of styling an element specifies, using the standard CSS/SVG styling model. The **content** of the element supplies the value of the styling parameter. The value may contain *expressions*.

The following parameters are supported:

Parameter	Required?	Description
`name="stroke"`	No	Specifies the solid color given to the line, in the form `#RRGGBB`. Default is black (`#000000`).
`name="stroke-width"`	No	Specifies the width of the line in pixels. Default is `1`.
`name="stroke-opacity"`	No	Specifies the opacity (transparency) of the line. The value is a number are between `0` (completely transparent) and `1` (completely opaque). Default is `1`.
`name="stroke-linejoin"`	No	Determines how lines are rendered at intersections of line segments. Possible values are `mitre` (sharp corner), `round` (rounded corner), and `bevel` (diagonal corner). Default is `mitre`.
`name="stroke-linecap"`	No	Determines how lines are rendered at their ends. Possible values are `butt` (sharp square edge), `round` (rounded edge), and `square` (slightly elongated square edge). Default is `butt`.
`name="stroke-dasharray"`	No	Encodes a dash pattern as a series of numbers separated by spaces. Odd-indexed numbers (first, third, etc) determine the length in pxiels to draw the line, and even-indexed numbers (second, fourth, etc) determine the length in pixels to blank out the line. Default is an unbroken line. *Starting from version 2.1* dash arrays can be combined with graphic strokes to generate complex line styles with alternating symbols or a mix of lines and symbols.
`name="stroke-dashoffset"`	No	Specifies the distance in pixels into the `dasharray` pattern at which to start drawing. Default is `0`.

PerpendicularOffset

The `<PerpendicularOffset>` element is optional. It is native to the SE 1.1 specification, but supported also in SLD 1.0 as a vendor extension.

If present, it makes the renderer draw a line parallel to the original one, at the given distance. When applied on lines, positive values generate a parallel line on the left hand side, negative values on the right hand side. When applied on polygons instead, positive is interpreted as outwards, negative as inwards.

As most properties, `<PerpendicularOffset>` accepts expressions.

Care should be taken when using it, as it might become a performance bottleneck. When offsetting lines a fast offset algorithm is used, which works well at small distances, but can generate visible artifacts at higher values. When working against polygons the fast offset line algorithm is used up to 3 pixels away from the original geometry, after that a full buffer algorithm is used instead, which always provides correct results, but is significantly more expensive.

Basic Example

The following symbolizer is taken from the *Lines* section in the *SLD Cookbook*.

```
      <LineSymbolizer>
        <Stroke>
          <CssParameter name="stroke">#0000FF</CssParameter>
          <CssParameter name="stroke-width">3</CssParameter>
          <CssParameter name="stroke-dasharray">5 2</CssParameter>
        </Stroke>
      </LineSymbolizer>
```

The symbolizer styles a feature as a dashed blue line of width 3 pixels.

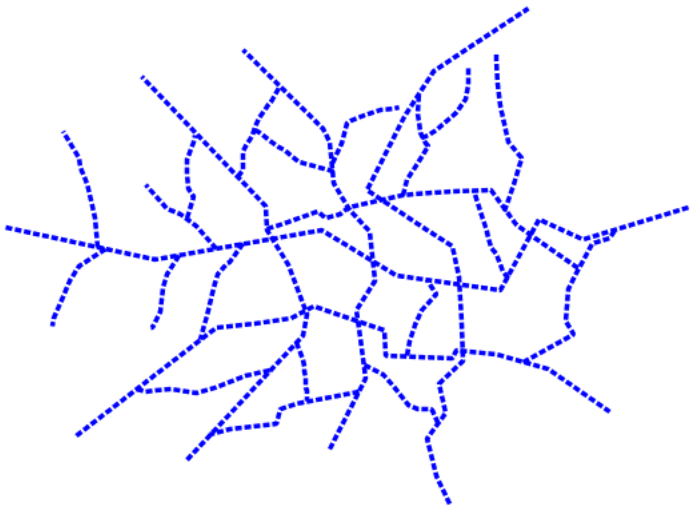


Fig. 6.70: *Dashed blue line*

Offsetting lines

The following style excerpt generates a solid line, and then a dashed blue line 3 pixels on the left of it.

```
      <LineSymbolizer>
        <Stroke>
          <CssParameter name="stroke">#000000</CssParameter>
          <CssParameter name="stroke-width">2</CssParameter>
        </Stroke>
      </LineSymbolizer>
      <LineSymbolizer>
        <Stroke>
          <CssParameter name="stroke">#0000FF</CssParameter>
          <CssParameter name="stroke-width">3</CssParameter>
          <CssParameter name="stroke-dasharray">5 2</CssParameter>
        </Stroke>
        <PerpendicularOffset>3</PerpendicularOffset>
      </LineSymbolizer>
```

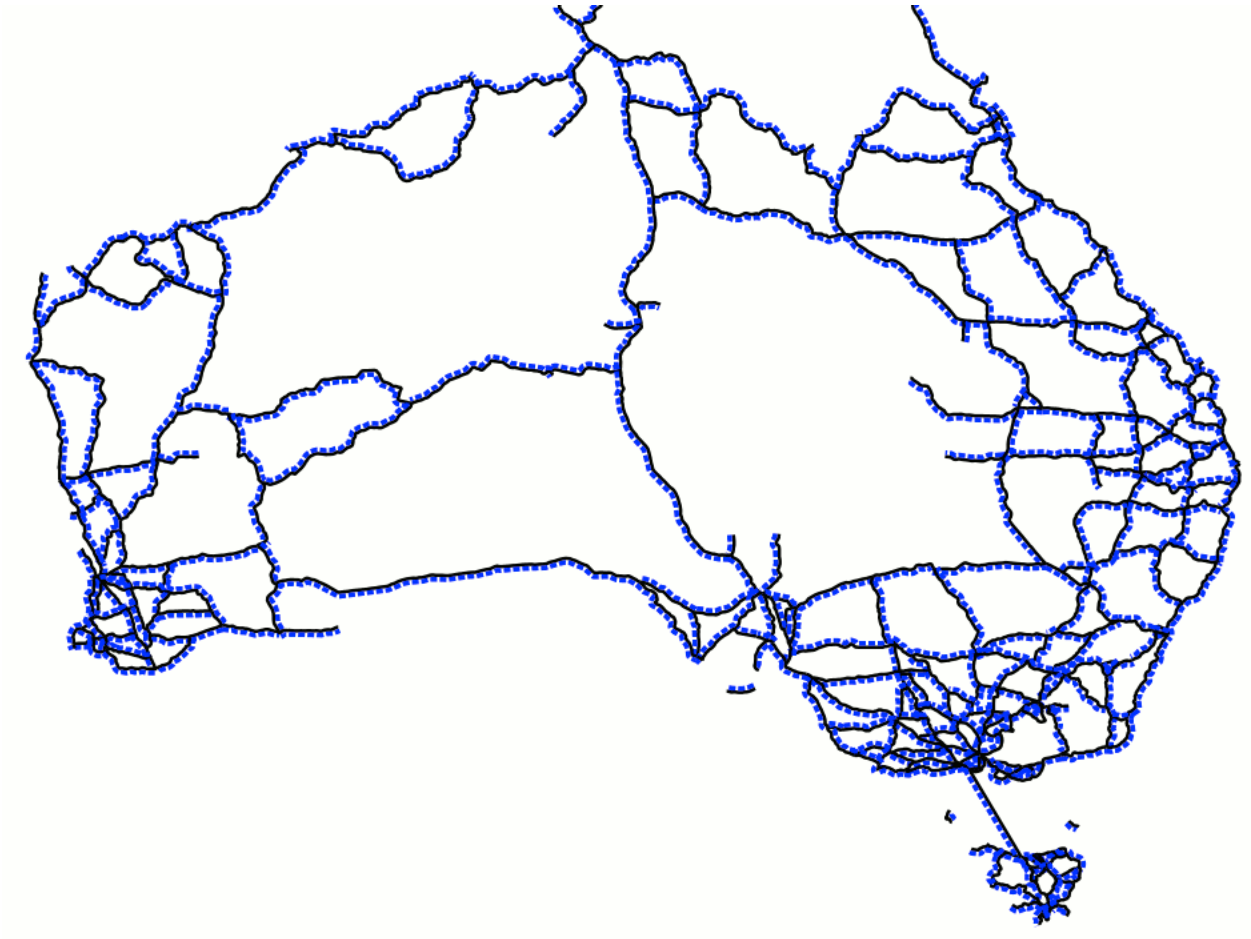


Fig. 6.71: *Left offset dashed line*

Offsetting polygons

The following style excerpt builds a inward offset line for polygons.

```
      <PolygonSymbolizer>
        <Stroke>
          <CssParameter name="stroke">#000000</CssParameter>
          <CssParameter name="stroke-width">2</CssParameter>
        </Stroke>
      </PolygonSymbolizer>
      <LineSymbolizer>
        <Stroke>
          <CssParameter name="stroke">#AAAAAA</CssParameter>
          <CssParameter name="stroke-width">3</CssParameter>
        </Stroke>
        <PerpendicularOffset>-2</PerpendicularOffset>
      </LineSymbolizer>
```

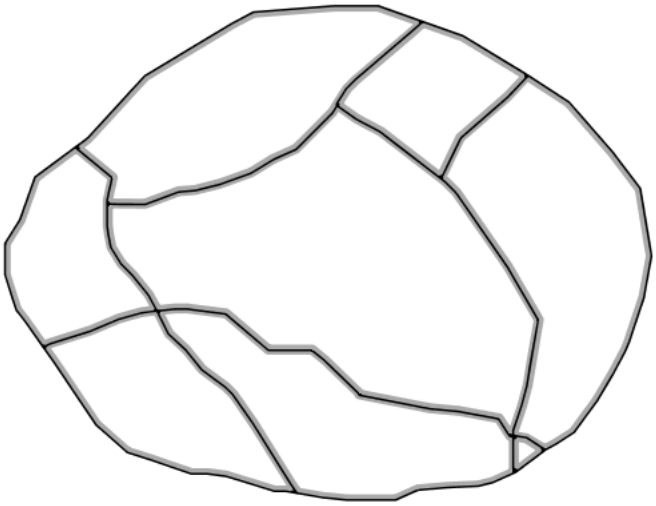


Fig. 6.72: *Inwards offset line*

PolygonSymbolizer

A **PolygonSymbolizer** styles features as **polygons**. Polygons are two-dimensional geometries. They can be depicted with styling for their interior (fill) and their border (stroke). Polygons may contain one or more holes, which are stroked but not filled. When rendering a polygon, the fill is rendered before the border is stroked.

Syntax

A `<PolygonSymbolizer>` contains an optional `<Geometry>` element, and two elements `<Fill>` and `<Stroke>` for specifying styling:

Tag	Required?	Description
`<Geometry>`	No	Specifies the geometry to be rendered.
`<Fill>`	No	Specifies the styling for the polygon interior.
`<Stroke>`	No	Specifies the styling for the polygon border.

Geometry

The `<Geometry>` element is optional. If present, it specifies the featuretype property from which to obtain the geometry to style using the `PropertyName` element. See also *Geometry transformations in SLD* for GeoServer extensions for specifying geometry.

Any kind of geometry may be styled with a `<PolygonSymbolizer>`. Point geometries are treated as small orthonormal square polygons. Linear geometries are closed by joining their ends.

Stroke

The `<Stroke>` element specifies the styling for the **border** of a polygon. The syntax is described in the `<LineSymbolizer>` *Stroke* section.

Fill

The `<Fill>` element specifies the styling for the **interior** of a polygon. It can contain the sub-elements:

Tag	Required?	Description
`<GraphicFill>`	No	Renders the fill of the polygon with a repeated pattern.
`<CssParameter>`	0..N	Specifies parameters for filling with a solid color.

GraphicFill

The `<GraphicFill>` element contains a `<Graphic>` element, which specifies a graphic image or symbol to use for a repeated fill pattern. The syntax is described in the `PointSymbolizer` *Graphic* section.

CssParameter

The `<CssParameter>` elements describe the styling of a solid polygon fill. Any number of `<CssParameter>` elements can be specified.

The `name` **attribute** indicates what aspect of styling an element specifies, using the standard CSS/SVG styling model. The **content** of the element supplies the value of the styling parameter. The value may contain *expressions*.

The following parameters are supported:

Parameter	Required?	Description
`name="fill"`	No	Specifies the fill color, in the form `#RRGGBB`. Default is grey (`#808080`).
`name="fill-opacity"`	No	Specifies the opacity (transparency) of the fill. The value is a decimal number between `0` (completely transparent) and `1` (completely opaque). Default is `1`.

Example

The following symbolizer is taken from the *Polygons* section in the *SLD Cookbook*.

```
<PolygonSymbolizer>
  <Fill>
    <CssParameter name="fill">#000080</CssParameter>
  </Fill>
</PolygonSymbolizer>
```

This symbolizer contains only a `<Fill>` element. Inside this element is a `<CssParameter>` that specifies the fill color for the polygon to be `#000080` (a muted blue).

Further examples can be found in the *Polygons* section of the *SLD Cookbook*.

TextSymbolizer

A **TextSymbolizer** styles features as **text labels**. Text labels are positioned eoither at points or along linear paths derived from the geometry being labelled.

Labelling is a complex operation, and effective labelling is crucial to obtaining legible and visually pleasing cartographic output. For this reason SLD provides many options to control label placement. To improve quality even more GeoServer provides additional options and parameters. The usage of the standard and extended options are described in greater detail in the following section on *Labeling*.

Syntax

A `<TextSymbolizer>` contains the following elements:

Tag	Required?	Description
`<Geometry>`	No	The geometry to be labelled.
`<Label>`	No	The text content for the label.
`<Font>`	No	The font information for the label.
`<LabelPlacement>`	No	Sets the position of the label relative to its associated geometry.
`<Halo>`	No	Creates a colored background around the label text, for improved legibility.
`<Fill>`	No	The fill style of the label text.
`<Graphic>`	No	A graphic to be displayed behind the label text. See *Graphic* for content syntax.
`<Priority>`	No	The priority of the label during conflict resolution. Content may contains *expressions*. See also *Priority Labeling*.
`<VendorOption>`	0..N	A GeoServer-specific option. See *Labeling* for descriptions of the available options. Any number of options may be specified.

Geometry

The `<Geometry>` element is optional. If present, it specifies the featuretype property from which to obtain the geometry to label, using a `<PropertyName>` element. See also *Geometry transformations in SLD* for GeoServer extensions for specifying geometry.

Any kind of geometry may be labelled with a `<TextSymbolizer>`. For non-point geometries, a representative point is used (such as the centroid of a line or polygon).

Label

The `<Label>` element specifies the text that will be rendered as the label. It allows content of mixed type, which means that the content can be a mixture of string data and *Filter Expressions*. These are concatenated to form the final label text. If a label is provided directly by a feature property, the content is a single `<PropertyName>`. Multiple properties can be included in the label, and property values can be manipulated by filter expressions and functions. Additional "boilerplate" text can be provided as well. Whitespace can be preserved by surrounding it with XML `<![CDATA[ ]]>` delimiters.

If this element is omitted, no label is rendered.

Font

The `<Font>` element specifes the font to be used for the label. A set of `<CssParameter>` elements specify the details of the font.

The `name` **attribute** indicates what aspect of the font is described, using the standard CSS/SVG font model. The **content** of the element supplies the value of the font parameter. The value may contain *expressions*.

Parameter	Required?	Description
`name="font-family"`	No	The family name of the font to use for the label. Default is `Times`.
`name="font-style"`	No	The style of the font. Options are `normal`, `italic`, and `oblique`. Default is `normal`.
`name="font-weight"`	No	The weight of the font. Options are `normal` and `bold`. Default is `normal`.
`name="font-size"`	No	The size of the font in pixels. Default is `10`.

LabelPlacement

The `<LabelPlacement>` element specifies the placement of the label relative to the geometry being labelled. There are two possible sub-elements: `<PointPlacement>` or `<LinePlacement>`. Exactly one of these must be specified.

Tag	Required?	Description
`<PointPlacement>`	No	Labels a geometry at a single point
`<LinePlacement>`	No	Labels a geometry along a linear path

PointPlacement

The `<PointPlacement>` element indicates the label is placed at a labelling point derived from the geometry being labelled. The position of the label relative to the labelling point may be controlled by the following sub-elements:

Tag	Required?	Description
`<AnchorPoint>`	No	The location within the label bounding box that is aligned with the label point. The location is specified by `<AnchorPointX>` and `<AnchorPointY>` sub-elements, with values in the range [0..1]. Values may contain *expressions*.
`<Displacement>`	No	Specifies that the label point should be offset from the original point. The offset is specified by `<DisplacementX>` and `<DisplacementY>` sub-elements, with values in pixels. Values may contain *expressions*. Default is `(0, 0)`.
`<Rotation>`	No	The rotation of the label in clockwise degrees (negative values are counterclockwise). Value may contain *expressions*. Default is `0`.

The anchor point justification, displacement offsetting, and rotation are applied in that order.

LinePlacement

The `<LinePlacement>` element indicates the label is placed along a linear path derived from the geometry being labelled. The position of the label relative to the linear path may be controlled by the following sub-element:

Tag	Required?	Description
`<PerpendicularOffset>`	No	The offset from the linear path, in pixels. Positive values offset to the left of the line, negative to the right. Value may contain *expressions*. Default is `0`.

The appearance of text along linear paths can be further controlled by the vendor options `followLine`, `maxDisplacement`, `repeat`, `labelAllGroup`, and `maxAngleDelta`. These are described in *Labeling*.

Halo

A halo creates a colored background around the label text, which improves readability in low contrast situations. Within the `<Halo>` element there are two sub-elements which control the appearance of the halo:

Tag	Required?	Description
`<Radius>`	No	The halo radius, in pixels. Value may contain *expressions*. Default is `1`.
`<Fill>`	No	The color and opacity of the halo via `CssParameter` elements for `fill` and `fill-opacity`. See *Fill* for full syntax. The parameter values may contain *expressions*. Default is a **white** fill (`#FFFFFF`) at **100%** opacity.

Fill

The `<Fill>` element specifies the fill style for the label text. The syntax is the same as that of the `PolygonSymbolizer` *Fill* element. The default fill color is **black** (`#FFFFFF`) at **100%** opacity..

Graphic

The `<Graphic>` element specifies a graphic symbol to be displayed behind the label text (if any). A classic use for this is to display "highway shields" behind road numbers provided by feature attributes. The element content has the same syntax as the `<PointSymbolizer>` *Graphic* element. Graphics can be provided by internal *mark symbols*, or by external images or SVG files. Their size and aspect ratio can be changed to match the text displayed with them by using the vendor options *graphic-resize* and *graphic-margin*.

Example

The following symbolizer is taken from the *Points* section in the *SLD Cookbook*.

```
      <TextSymbolizer>
        <Label>
          <ogc:PropertyName>name</ogc:PropertyName>
        </Label>
        <Font>
          <CssParameter name="font-family">Arial</CssParameter>
          <CssParameter name="font-size">12</CssParameter>
          <CssParameter name="font-style">normal</CssParameter>
          <CssParameter name="font-weight">bold</CssParameter>
        </Font>
        <LabelPlacement>
          <PointPlacement>
            <AnchorPoint>
              <AnchorPointX>0.5</AnchorPointX>
              <AnchorPointY>0.0</AnchorPointY>
            </AnchorPoint>
            <Displacement>
              <DisplacementX>0</DisplacementX>
              <DisplacementY>25</DisplacementY>
            </Displacement>
            <Rotation>-45</Rotation>
          </PointPlacement>
        </LabelPlacement>
        <Fill>
          <CssParameter name="fill">#990099</CssParameter>
        </Fill>
      </TextSymbolizer>
```

The symbolizer labels features with the text from the `name` property. The font is Arial in bold at 12 pt size, filled in purple. The labels are centered on the point along their lower edge, then displaced 25 pixels upwards, and finally rotated 45 degrees counterclockwise.

The displacement takes effect before the rotation during rendering, so the 25 pixel vertical displacement is itself rotated 45 degrees.

Scalable Font Size

The font size can also be set depending on the scale denominator as follows:

```
      <CssParameter name="font-size">
        <ogc:Function name="Categorize">
          <!-- Value to transform -->
          <ogc:Function name="env">
```

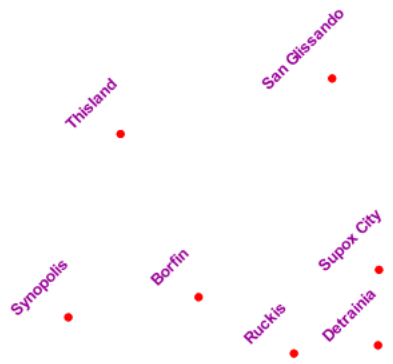


Fig. 6.73: *Point with rotated label*

```
            <ogc:Literal>wms_scale_denominator</ogc:Literal>
          </ogc:Function>
          <!-- Output values and thresholds -->
          <!-- Ranges: -->
          <!-- [scale <= 300, font 12] -->
          <!-- [scale 300 - 2500, font 10] -->
          <!-- [scale > 2500, font 8] -->
          <ogc:Literal>12</ogc:Literal>
          <ogc:Literal>300</ogc:Literal>
          <ogc:Literal>10</ogc:Literal>
          <ogc:Literal>2500</ogc:Literal>
          <ogc:Literal>8</ogc:Literal>
        </ogc:Function>
      </CssParameter>
```

The above example would display text at different sizes depending on the scale denominator setting. A font size of **12** for scale denominator of less than or equal to 300, a font size of **10** for scale denominator from 300-2500 and a font size of **8** for scale denominator greater than 2500.

Labeling

This section discusses the details of controlling label placement via the standard SLD options. It also describes a number of GeoServer enhanced options for label placement that provide better cartographic output.

LabelPlacement

The SLD specification defines two alternative label placement strategies which can be used in the `<LabelPlacement>` element:

- `<PointPlacement>` places labels at a single point
- `<LinePlacement>` places labels along a line

PointPlacement

When `<PointPlacement>` is used the geometry is labelled at a single **label point**. For lines, this point lies at the middle of the visible portion of the line. For polygons, the point is the centroid of the visible portion of the polygon. The position of the label relative to the label point can be controlled by the following sub-elements:

Element	Description
`<AnchorPoint>`	Determines the placement of the label relative to the label point. Values given as decimals between 0-1.
`<Displacement>`	Offsets the label from the anchor point. Values given in pixels.
`<Rotation>`	Rotates the label clockwise by a given number of degrees.

The best way to explain these options is with examples.

AnchorPoint

The anchor point determines where the label is placed relative to the label point.

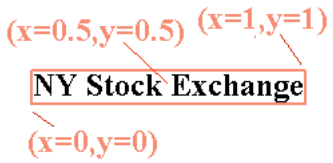
```
<AnchorPoint>
  <AnchorPointX>
    0.5
  </AnchorPointX>
  <AnchorPointY>
    0.5
  </AnchorPointY>
</AnchorPoint>
```

The anchor point values—listed here as (X, Y) ordered pairs—are specified relative to the bounding box of the label, with values from 0 to 1 inclusive. For example:

- *(Default)* Bottom left of the box is (0, 0)
- Top right is (1, 1)
- Center is (0.5, 0.5)

So to have the anchor location centered just below the label (label top-centered), use (0.5, 0):

```
<AnchorPoint>
  <AnchorPointX>
    0.5
  </AnchorPointX>
  <AnchorPointY>
    0
  </AnchorPointY>
</AnchorPoint>
```



The following examples show how changing the anchor point affects the position of labels:

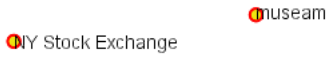


Fig. 6.74: (0, 0.5) places the label to the right of the label point

Fig. 6.75: (0.5, 0.5) places the center of the label at the label point

Displacement

Displacement allows fine control of the placement of the label. The displacement values offset the location of the label from the anchor point by a specified number of pixels. The element syntax is:

```
<Displacement>
  <DisplacementX>
     10
  </DisplacementX>
  <DisplacementY>
      0
  </DisplacementY>
</Displacement>
```

Examples:

Displacement of X=10 pixels (compare with default anchor point of (X=0, Y=0.5) shown above)

Displacement of Y=-10 pixels (compare with anchor point (X= 0.5, Y=1.0) - not shown)

Rotation

The optional `<Rotation>` element specifies that labels should be rotated clockwise by a given number of degrees

```
<Rotation>
  45
</Rotation>
```

The examples below show how the rotation interacts with anchor points and displacements.

45 degree rotation

45 degree rotation with anchor point (X=0.5, Y=0.5)

Fig. 6.76: (1, 0.5) places the label to the left of the label point

Fig. 6.77: (0.5, 0) places the label horizontally centered above the label point

45 degree rotation with 40-pixel X displacement

45 degree rotation with 40-pixel Y displacement with anchor point (X=0.5, Y=0.5)

LinePlacement

To label linear features (such as a road or river), the `<LinePlacement>` element can be specified. This indicates that the styler should determine the best placement and rotation for the labels along the lines.

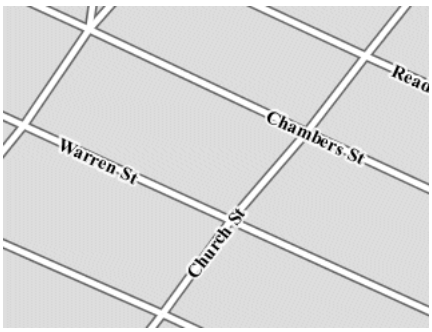
The standard SLD LinePlacement element provides one optional sub-element, `<PerpendicularOffset>`. GeoServer provides much more control over line label placement via vendor-specific options; see below for details.

PerpendicularOffset

The optional `<PerpendicularOffset>` element allows you to position a label above or below a line. (This is similiar to the `<DisplacementY>` for label points described above.) The displacement value is specified in pixels. A positive value displaces upwards, a negative value downwards.

```
<LabelPlacement>
  <LinePlacement>
    <PerpendicularOffset>
       10
    </PerpendicularOffset>
  </LinePlacement>
</LabelPlacement>
```

Examples:



PerpendicularOffset = 0 (default)

PerpendicularOffset = 10

Composing labels from multiple attributes

The `<Label>` element in *<TextSymbolizer>* allows mixed content. This means its content can be a mixture of plain text and *Filter Expressions*. The mix gets interepreted as a concatenation. You can leverage this to create complex labels out of multiple attributes.

For example, if you want both a state name and its abbreviation to appear in a label, you can do the following:

```
<Label>
  <ogc:PropertyName>STATE_NAME</ogc:PropertyName> (<ogc:PropertyName>STATE_ABBR</
↪ogc:PropertyName>)
</Label>
```

and you'll get a label looking like `Texas (TX)`.

If you need to add extra white space or newline, you'll stumble into an XML oddity. The whitespace handling in the Label element is following a XML rule called "collapse", in which all leading and trailing whitespaces have to be removed, whilst all whitespaces (and newlines) in the middle of the xml element are collapsed into a single whitespace.

So, what if you need to insert a newline or a sequence of two or more spaces between your property names? Enter CDATA. CDATA is a special XML section that has to be returned to the interpreter as-is, without following any whitespace handling rule. So, for example, if you wanted to have the state abbreviation sitting on the next line you'd use the following:

```
<Label>
  <ogc:PropertyName>STATE_NAME</ogc:PropertyName><![CDATA[
]]>(<ogc:PropertyName>STATE_ABBR</ogc:PropertyName>)
</Label>
```

GeoServer Enhanced Options

GeoServer provides a number of label styling options as extensions to the SLD specification. Using these options gives more control over how the map looks, since the SLD standard isn't expressive enough to provide all the options one might want.

These options are specified as subelements of `<TextSymbolizer>`.

Priority Labeling

The optional `<Priority>` element allows specifying label priority. This controls how conflicts (overlaps) between labels are resolved during rendering. The element content may be an *expression* to retrieve or calculate a relative priority value for each feature in a layer. Alternatively, the content may be a constant value, to set the priority of a layer's labels relative to other layers on a rendered map.

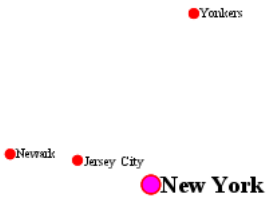
The default priority for labels is 1000.

Note: Standard SLD Conflict Resolution

If the `<Priority>` element is not present, or if a group of labels all have the same priority, then standard SLD label conflict resolution is used. Under this strategy, the label to display out of a group of conflicting labels is chosen essentially at random.

For example, take the following dataset of cities:

```
City Name    | population
-------------+-------------
Yonkers      |     197,818
Jersey City  |     237,681
Newark       |     280,123
New York     |   8,107,916
```



City locations (large scale map)

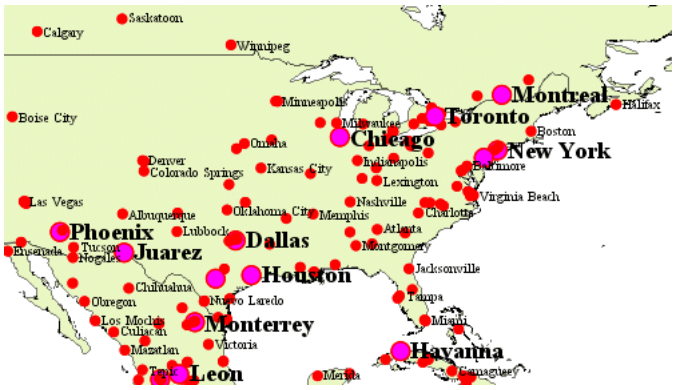
More people know where New York City is than where Jersey City is. Thus we want to give the label "New York" priority so it will be visible when in conflict with (overlapping) "Jersey City". To do this we include the following code in the `<TextSymbolizer>`:

```
<Priority>
    <ogc:PropertyName>population</ogc:PropertyName>
</Priority>
```

This ensures that at small scales New York is labeled in preference to the less populous cities nearby:

City locations (small scale map)

Without priority labeling, Jersey City could be labeled in preference to New York, making it difficult to interpret the map. At scales showing many features, priority labeling is essential to ensure that larger cities are more visible than smaller cities.



Grouping Features (group)

The `group` option allows displaying a single label for multiple features in a logical group.

```
<VendorOption name="group">yes</VendorOption>
```

Grouping works by collecting all features with the same label text, then choosing a representative geometry for the group, according to the following rules:

Geometry	Label Point
Point Set	The first point inside the view rectangle is used.
Line Set	Lines are joined together, clipped to the view rectangle, and the longest path is used.
Polygon Set	Polygons are clipped to the view rectangle, and the largest polygon is used.

If desired the labeller can be forced to label every element in a group by specifying the *labelAllGroup* option.

Warning: Be careful that the labels truly indicate features that should be grouped together. For example, grouping on city name alone might end up creating a group containing both *Paris* (France) and *Paris* (Texas).

Road data is a classic example to show why grouping is useful. It is usually desirable to display only a single label for all of "Main Street", not a label for every block of "Main Street."

When the `group` option is off (the default), grouping is not performed and every block feature is labeled (subject to label deconfliction):

When the `group` option is used, geometries with the same label are grouped together and the label position is determined from the entire group. This produces a much less cluttered map:

labelAllGroup

The `labelAllGroup` option can be used in conjunction with the `group` option (see *Grouping Features (group)*). It causes *all* of the disjoint paths in a line group to be labeled, not just the longest one.

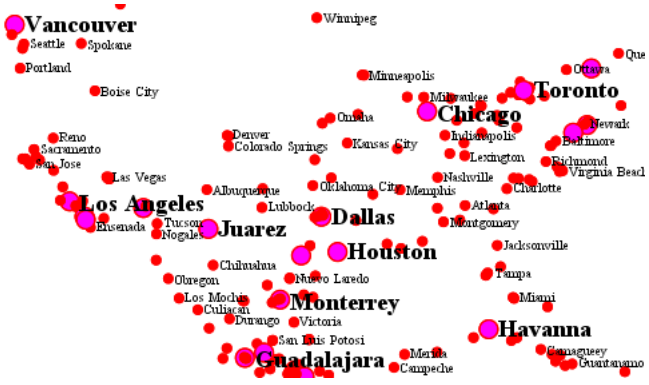
```
<VendorOption name="labelAllGroup">true</VendorOption>
```

Overlapping and Separating Labels (spaceAround)

By default GeoServer will not render labels "on top of each other". By using the `spaceAround` option you can either allow labels to overlap, or add extra space around labels. The value supplied for the option is a positive or negative size, in pixels.

```
<VendorOption name="spaceAround">10</VendorOption>
```

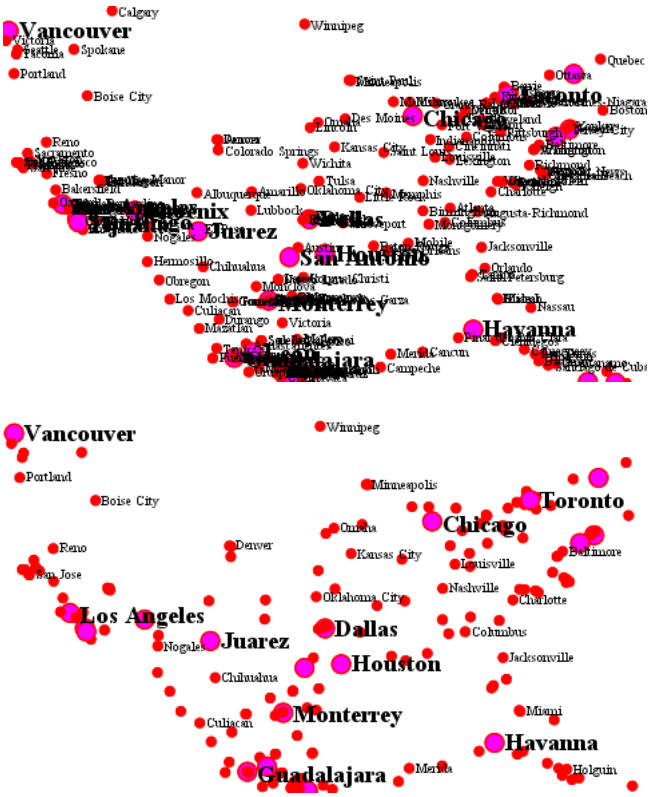
Using the default value of 0, the bounding box of a label cannot overlap the bounding box of another label:



With a negative `spaceAround` value, overlapping is allowed:

With a positive `spaceAround` value of 10, each label is at least 20 pixels apart from others:

Positive `spaceAround` values actually provide twice the space that you might expect. This is because you can specify a spaceAround for one label as 5, and for another label (in another TextSymbolizer) as 3. The



total distance between them is 8. Two labels in the first symbolizer ("5") will each be 5 pixels apart from each other, for a total of 10 pixels.

Note: Interaction between values in different TextSymbolizers

You can have multiple TextSymbolizers in your SLD file, each with a different `spaceAround` option. If all the `spaceAround` options are >=0, this will do what you would normally expect. If you have negative values ('allow overlap') then these labels can overlap labels that you've said should not be overlapping. If you don't like this behavior, it's not difficult to change - feel free to submit a patch!

---

followLine

The `followLine` option forces a label to follow the curve of the line. To use this option add the following to the `<TextSymbolizer>`.

Note: Straight Lines

You don't need to use followLine for straight lines. GeoServer will automatically follow the orientation of the line. However in this case `followLine` can be used to ensure the text isn't rendered if longer than the line.

```
<VendorOption name="followLine">true</VendorOption>
```

It is required to use `<LinePlacement>` along with this option to ensure that labels are placed along lines:

```
<LabelPlacement>
  <LinePlacement/>
</LabelPlacement>
```

maxDisplacement

The `maxDisplacement` option controls the displacement of the label along a line, around a point and inside a polygon.

For lines, normally GeoServer labels a line at its center point only. If this label conflicts with another one it may not be displayed at all. When this option is enabled the labeller will attempt to avoid conflic by using an alternate location within **maxDisplacement** pixels along the line from the pre-computed label point.

If used in conjunction with *repeat*, the value for `maxDisplacement` should always be **lower** than the value for `repeat`.

For points this causes the renderer to start circling around the point in search of a empty stop to place the label, step by step increasing the size of the circle until the max displacement is reached. The same happens for polygons, around the polygon labelling point (normally the centroid).

```
<VendorOption name="maxDisplacement">10</VendorOption>
```

repeat

The `repeat` option determines how often GeoServer displays labels along a line. Normally GeoServer labels each line only once, regardless of length. Specifying a positive value for this option makes the labeller attempt to draw the label every **repeat** pixels. For long or complex lines (such as contour lines) this makes labeling more informative.

```
<VendorOption name="repeat">100</VendorOption>
```

maxAngleDelta

When used in conjunction with *followLine*, the `maxAngleDelta` option sets the maximum angle, in degrees, between two subsequent characters in a curved label. Large angles create either visually disconnected words or overlapping characters. It is advised not to use angles larger than 30.

```
<VendorOption name="maxAngleDelta">15</VendorOption>
```

autoWrap

The `autoWrap` option wraps labels when they exceed the given width (in pixels). The size should be wide enough to accommodate the longest word, otherwise single words will be split over multiple lines.

```
<VendorOption name="autoWrap">50</VendorOption>
```

Labeling with autoWrap enabled

forceLeftToRight

The renderer tries to draw labels along lines so that the text is upright, for maximum legibility. This means a label may not follow the line orientation, but instead may be rotated 180° to display the text the right way up. In some cases altering the orientation of the label is not desired; for example, if the label is a directional arrow showing the orientation of the line.

The `forceLeftToRight` option can be set to `false` to disable label flipping, making the label always follow the inherent orientation of the line being labelled:

```
<VendorOption name="forceLeftToRight">false</VendorOption>
```

conflictResolution

By default labels are subject to **conflict resolution**, meaning the renderer will not allow any label to overlap with a label that has been already drawn. Setting the `conflictResolution` option to `false` causes this label to bypass conflict resolution. This means the label will be drawn even if it overlaps with other labels, and other labels drawn after it may overlap it.

```
<VendorOption name="conflictResolution">false</VendorOption>
```

goodnessOfFit

GeoServer will remove labels if they are a particularly bad fit for the geometry they are labeling.

Geometry	Goodness of Fit Algorithm
Point	Always returns 1.0 since the label is at the point
Line	Always returns 1.0 since the label is always placed on the line.
Polygon	The label is sampled approximately at every letter. The distance from these points to the polygon is determined and each sample votes based on how close it is to the polygon. (see LabelCacheDefault#goodnessOfFit())

The default value is 0.5, but it can be modified using:

```
<VendorOption name="goodnessOfFit">0.3</VendorOption>
```

polygonAlign

GeoServer normally tries to place labels horizontally within a polygon, and gives up if the label position is busy or if the label does not fit enough in the polygon. This option allows GeoServer to try alternate rotations for the labels.

```
<VendorOption name="polygonAlign">mbr</VendorOption>
```

Option	Description
`manual`	The default value. Only a rotation manually specified in the `<Rotation>` tag will be used
`ortho`	If the label does not fit horizontally and the polygon is taller than wider then vertical alignment will also be tried
`mbr`	If the label does not fit horizontally the minimum bounding rectangle will be computed and a label aligned to it will be tried out as well

graphic-resize

When a `<Graphic>` is specified for a label by default it is displayed at its native size and aspect ratio. The `graphic-resize` option instructs the renderer to magnify or stretch the graphic to fully contain the text of the label. If this option is used the `graphic-margin` option may also be specified.

```
<VendorOption name="graphic-resize">stretch</VendorOption>
```

Option	Description
`none`	Graphic is displayed at its native size (default)
`proportional`	Graphic size is increased uniformly to contain the label text
`stretch`	Graphic size is increased anisotropically to contain the label text

Labeling with a Graphic Mark "square" - L) at native size; R) with "graphic-resize"=stretch and "graphic-margin"=3

graphic-margin

The `graphic-margin` options specifies a margin (in pixels) to use around the label text when the `graphic-resize` option is specified.


```
<VendorOption name="graphic-margin">margin</VendorOption>
```

partials

The `partials` options instructs the renderer to render labels that cross the map extent, which are normally not painted since there is no guarantee that a map put on the side of the current one (tiled rendering) will contain the other half of the label. By enabling "partials" the style editor takes responsibility for the other half being there (maybe because the label points have been placed by hand and are assured not to conflict with each other, at all zoom levels).

```
<VendorOption name="partials">true</VendorOption>
```

underlineText

The `underlineText` option instruct the renderer to underline labels. The underline will work like a typical word processor text underline. The thickness and position of the underline will be defined by the font and color will be the same as the text. Spaces will also be underlined.

```
<VendorOption name="underlineText">true</VendorOption>
```

Some underlines examples:

strikethroughText

The `strikethroughText` option instruct the renderer to strikethrough labels. The strikethrough will work like a typical word processor text strikethrough. The thickness and position of the line will be defined by the font and color will be the same as the text. Spaces will also be stroken.

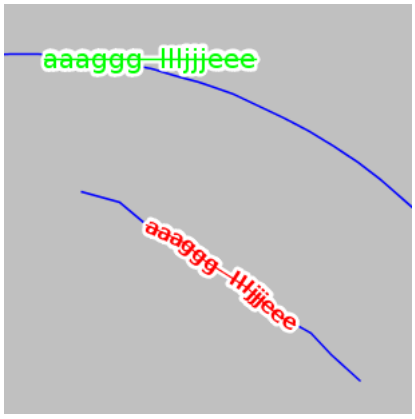
```
<VendorOption name="strikethroughText">true</VendorOption>
```

Some strikethrough examples:

charSpacing

The `charSpacing` option controls the amount of space between characters, a positive value increases it, a negative value shrinks it (and will eventually make characters overlap). The value is specified in pixels.


Non-primary A-road
Primary-A-road
Motorway
Primary-A-road
B
road
Non-primary A-road


aaaggg_llljjjeee
aaaggg_llljjjeee

```
<VendorOption name="charSpacing">3</VendorOption>
```

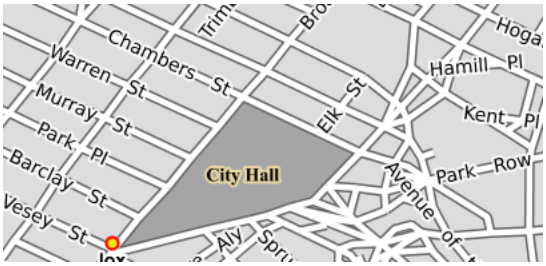
Example of adding 3 extra pixels of space between chars on road names:

wordSpacing

The `wordSpacing` option controls the amount of space between words, for this option only positive values (or zero) are accepted. The value is specified in pixels.

```
<VendorOption name="wordSpacing">5</VendorOption>
```

Example of adding 5 extra pixels of space between words on road names:



displacementMode

Comma separated list of label displacement directions for point/polygon labels (used along with maxDisplacement). The indicated directions will be tried in turn. Valid values are cardinal directions abbreviations, in particular, `N, W, E, S, NW, NE, SW, SE`.

The following example sets the typical "diagonal displacement" typically used for points:

```
<VendorOption name="displacementMode">NE, NW, SW, SE</VendorOption>
```

While this one allows displacement only in the vertical direction:

```
<VendorOption name="displacementMode">N, S</VendorOption>
```

RasterSymbolizer

GeoServer supports the ability to display raster data in addition to vector data.

Raster data is not merely a picture, rather it can be thought of as a grid of georeferenced information, much like a graphic is a grid of visual information (with combination of reds, greens, and blues). Unlike graphics,

which only contain visual data, each point/pixel in a raster grid can have many different attributes (bands), with possibly none of them having an inherently visual component.

With the above in mind, one needs to choose how to visualize the data, and this, like in all other cases, is done by using an SLD. The analogy to vector data is evident in the naming of the tags used. Vectors, consisting of points, line, and polygons, are styled by using the `<PointSymbolizer>`, `<LineSymbolizer>`, and `<PolygonSymbolizer>` tags. It is therefore not very surprising that raster data is styled with the tag <RasterSymbolizer>.

Syntax

The following elements can be used inside the `<RasterSymbolizer>` element.

- <Opacity>
- <ColorMap>
- <ChannelSelection>
- <ContrastEnhancement>
- <ShadedRelief> *
- <OverlapBehavior> *
- <ImageOutline> *

Warning: The starred (*) elements are not yet implemented in GeoServer.

Opacity

The `<Opacity>` element sets the transparency level for the entire rendered image. As is standard, the values range from zero (0) to one (1), with zero being transparent, and one being opaque. The syntax is:

```
<Opacity>0.5</Opacity>
```

where, in this case, the raster is rendered at 50% opacity.

ColorMap

The `<ColorMap>` element defines the color values for the pixels of a raster image, as either color gradients, or a mapping of specific values to fixed colors.

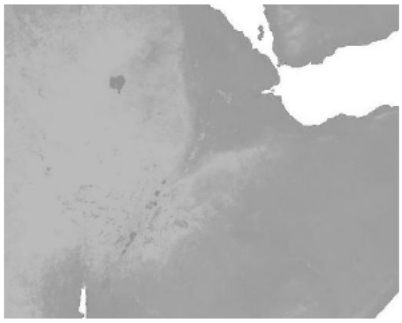
A color map is defined by a sequence of `<ColorMapEntry>` elements. Each `<ColorMapEntry>` element specifies a `color` and a `quantity` attribute. The quantity refers to the value of a raster pixel. The `color` value is denoted in standard hexadecimal RGB format (#RRGGBB). `<ColorMapEntry>` elements can also have `opacity` and `label` attributes. The `opacity` attribute overrides the global `<Opacity>` value. The `label` attribute is used to provide text for legends. A color map can contain up to 255 `<ColorMapEntry>` elements.

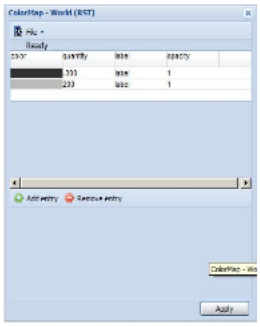
The simplest `<ColorMap>` has two color map entries. One specifyies a color for the "bottom" of the dataset, and the other specifyies a color for the "top" of the dataset. Pixels with values equal to or less than the minimum value are rendered with the bottom color (and opacity). Pixels with values equal to or great than the maximum value are rendered with the top color and opacity. The colors for values in between are automatically interpolated, making creating color gradients easy.

A color map can be refined by adding additional intermediate entries. This is useful if the dataset has discrete values rather than a gradient, or if a multi-colored gradient is desired. One entry is added for each different color to be used, along with the corresponding quantity value.

For example, a simple ColorMap can define a color gradient from color #323232 to color #BBBBBB over quantity values from -300 to 200:

```
<ColorMap>
        <ColorMapEntry color="#323232" quantity="-300" label="label1" opacity="1"/>
        <ColorMapEntry color="#BBBBBB" quantity="200" label="label2" opacity="1"/>
</ColorMap>
```

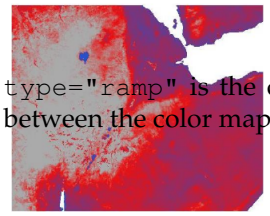


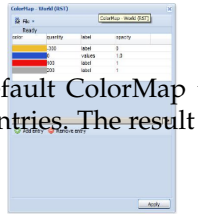


A more refined example defines a color gradient from color #FFCC32 through color #BBBBBB, running through color #3645CC and color #CC3636. The bottom color #FFCC32 is defined to be transparent This simulates an alpha channel, since pixels with values of -300 and below will not be rendered. Notice that the default opacity is 1 (opaque) when not specified.

GeoServer extends the `<ColorMap>` element to allow two attributes: `type` and `extended`.

type





The `<ColorMap>` `type` attribute specifies the kind of ColorMap to use. There are three different types of ColorMaps that can be specified: `ramp`, `intervals` and `values`.

`type="ramp"` is the default ColorMap type. It specifies that colors should be interpolated for values between the color map entries. The result is shown in the following example.

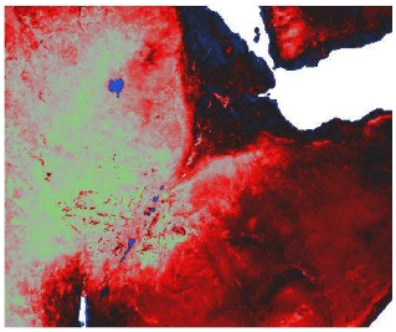
```
<ColorMap type="ramp">
        <ColorMapEntry color="#
↪" quantity="-300" label="labe
        <ColorMapEntry color="
↪#2851CC" quantity="0" label="
```

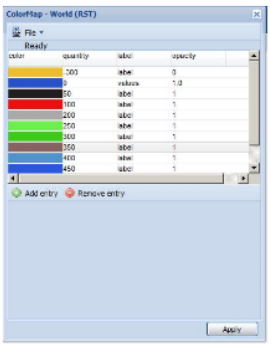
```
                <ColorMapEntry color="
↪#211F1F" quantity="50" label=
                <ColorMapEntry color="#
↪" quantity="100" label="label
                <ColorMapEntry color="#
↪" quantity="200" label="label
                <ColorMapEntry color="#
↪" quantity="250" label="label
                <ColorMapEntry color="#
↪" quantity="300" label="label
                <ColorMapEntry color="#
↪" quantity="350" label="label
                <ColorMapEntry color="#
↪" quantity="400" label="label
                <ColorMapEntry color="#
↪" quantity="450" label="label
                <ColorMapEntry color="#
↪" quantity="600" label="label
            </ColorMap>
```

`type="values"` means that only pixels with the specified entry quantity values are rendered. Pixels with other values are not rendered. Using the example set of color map entries:

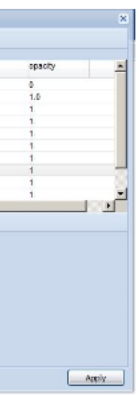
```
            <ColorMap type="values">
                <ColorMapEntry color="#
↪" quantity="-300" label="labe
                ...
                <ColorMapEntry color="#
↪" quantity="600" label="label
            </ColorMap>
```

The result image is:





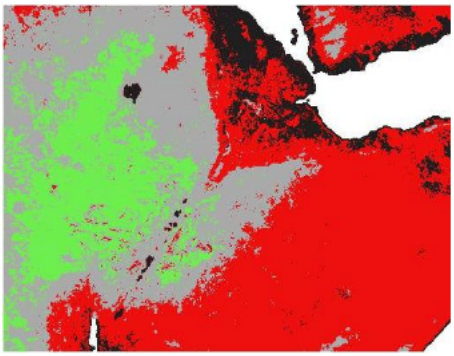
`type="intervals"` value means that each interval defined by two entries is rendered using the color of the first (lowest-value) entry. No color interpolation is applied across the intervals. Using the example set of color map entries:

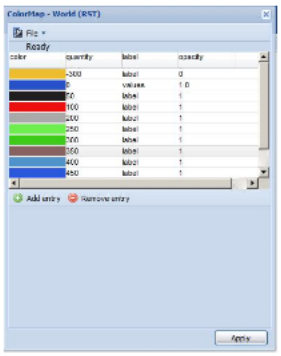


```
            <ColorMap type="intervals
↪" extended="true">
                <ColorMapEntry␣
↪color="#EEBE2F"␣
↪quantity="-300" label=
↪"label" opacity="0"/>
                ...
```

```
                <ColorMapEntry␣
↪color="#DDB02C
↪" quantity="600" label=
↪"label" opacity="1"/>
</ColorMap>
```

The result image is:

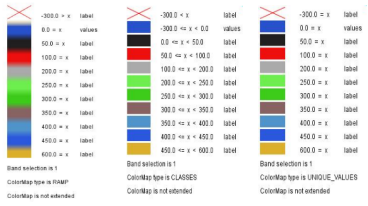




The color map type is also reflected in the legend graphic. A typical request for a raster legend is (using the `forceRule:true` option to force output of the color map):

```
↪OPTIONS=forceRule:true&
↪LAYER=it.
↪geosolutions:di08032_da
```

The legends returned for the different types are:



extended

The `extended` attribute specifies whether the color map gradient uses 256 (8-bit) or 65536 (16-bit) colors. The value `false` (the default) specifies that the color scale is calculated using 8-bit color, and `true` specifies using 16-bit color.

CQL Expressions

All of the ColorMapEntry attributes (color, quantity, label and opacity) can be defined using `cql expressions`, with the ${...expression...} syntax.

CQL expressions are useful to make the color map dynamic, using values taken from the client:

```
<ColorMapEntry color="#00FF00
↪" quantity="${env('low',3)}" label="Low" opacity="1"/>
<ColorMapEntry color="#FFFF00" quantity=
↪"${env('medium',10)}" label="Medium" opacity="1"/>
<ColorMapEntry color="#FF0000" quantity=
↪"${env('high',1000)}" label="High" opacity="1"/>
```

In this example quantity values are not fixed, but can be specified by the client using the ENV request parameter:

http://localhost:8080/geoserver/wms?REQUEST=GetMap&VERSION=1.0.0&...&ENV=low:10;medium:100;high:5

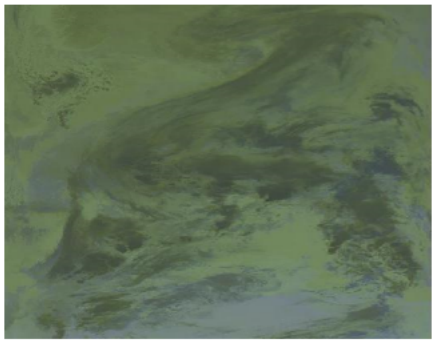
For a complete reference of CQL capabilities, see *here*

ChannelSelection

The `<ChannelSelection>` element specifies how dataset bands are mapped to image color channels. Named dataset bands may be mapped to red, green and blue channels, or a single named band may be mapped to a grayscale channel.

The following example maps source channels 1, 2 and 3 to the red, green, and blue color channels.

```
<ChannelSelection>
  <RedChannel>
    <SourceChannelName>1</SourceChannelName>
  </RedChannel>
  <GreenChannel>
    <SourceChannelName>2</SourceChannelName>
  </GreenChannel>
  <BlueChannel>
    <SourceChannelName>3</SourceChannelName>
  </BlueChannel>
</ChannelSelection>
```



The next example shows selecting a single band of an RGB image as a grayscale channel, and re-colorizing it via a ColorMap:

```
<RasterSymbolizer>
        <Opacity>1.0</Opacity>
        <ChannelSelection>
                <GrayChannel>
↪                   <SourceChannelName>1<
                  </GrayChannel>
        </ChannelSelection>
        <ColorMap extended="true">
↪   <ColorMapEntry color="#0000ff"
↪   <ColorMapEntry color="#009933"
↪ <ColorMapEntry color="#ff9900" c
↪ <ColorMapEntry color="#ff0000" c
        </ColorMap>
</RasterSymbolizer>
```

ChannelSelection Expressions

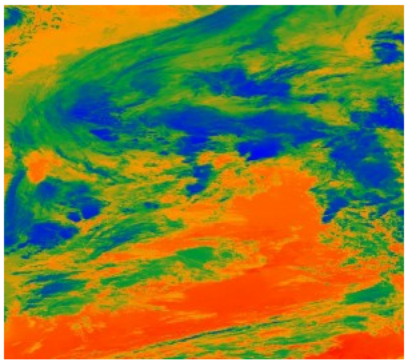
Since the previous approach supports Strings only and therefore is static and not suitable when dealing with multispectral imagery

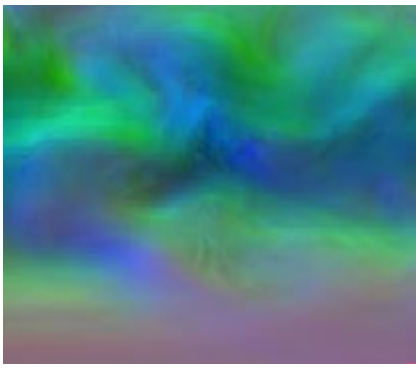
that has more than four bands and hyperspectral imagery (hyperspectral sensors have typically hundreds of bands), a dynamical approach is needed.

By replacing Strings with Expressions in `<SourceChannelName>`, context free functions like `env` can be used to indicate which bands are to be used in a particular rendering session.

The following example shows how to set the Red, Green and Blue channels and to map them into the desired bands. Here below, the `env` function will set, by default in the WMS request, the RedChannel on the second band, the GreenChannel on the fifth band and the BlueChannel on the seventh band.

```
<RasterSymbolizer>
<ChannelSelection>
    <RedChannel>
    <SourceChannelName>
            <ogc:Function name="env">
                    <ogc:Literal>B1</ogc:Literal>
                    <ogc:Literal>1</ogc:Literal>
            </ogc:Function>
    </SourceChannelName>
    </RedChannel>
    <GreenChannel>
    <SourceChannelName>
            <ogc:Function name="env">
                    <ogc:Literal>B2</ogc:Literal>
                    <ogc:Literal>2</ogc:Literal>
            </ogc:Function>
    </SourceChannelName>
    </GreenChannel>
    <BlueChannel>
    <SourceChannelName>
            <ogc:Function name="env">
                    <ogc:Literal>B3</ogc:Literal>
                    <ogc:Literal>3</ogc:Literal>
            </ogc:Function>
    </SourceChannelName>
    </BlueChannel>
</ChannelSelection>
<RasterSymbolizer>
```





The style Schema supports also the SLD 1.1 and CSS. As a CSS examples:

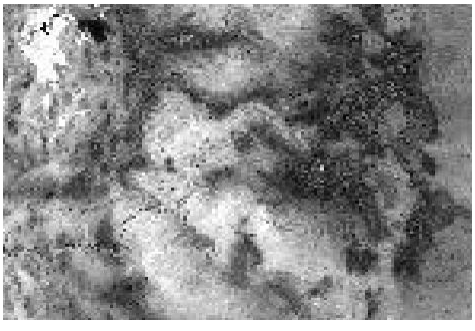
```
* { raster-channels: [env('B1','1')] '2

* { raster-channels: @B1(1)  '2' '3';}
```

One can specify the `env` request parameters in the WMS request to switch the bands and render the raster layer using the desired bands, for example the 4, 2, 3 as the following:

```
http://
↪localhost:8083/geosolutions/wms?servi
↪1.0&request=GetMap&layers=geosolution
↪multichannel&styles=&bbox=-180.
↪0,-90.5,180.0,90.5&width=768&height=3
↪format=application/openlayers&env=B1:
```

Now let us suppose that we want to work on a single band and to exclude all the remaining bands in order to render a monochromatic raster. As an SLD example:

```
<RasterSymbolizer>
  <Opacity>1.0</se:Opacity>
  <ChannelSelection>
    <GrayChannel>
      <SourceChannelName>
            <Function name="env">
             <ogc:Literal>B1</ogc:Liter
             <ogc:Literal>7</ogc:Litera
          </ogc:Function>
      </SourceChannelName>
    </GrayChannel>
  </ChannelSelection>
</RasterSymbolizer>
```

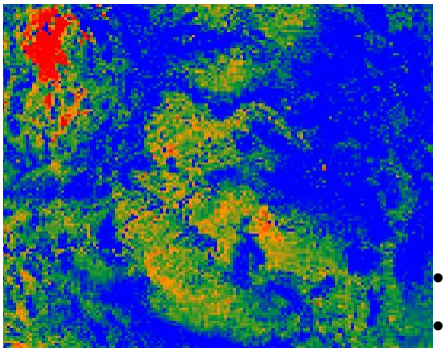


The Schema above will render the channel "7" by default. As before, you can choose to render any channel of the raster by calling the `env` function in your WMS request and setting the desired band. By adding to the request &env=B1:3 for example:

```
http://localhost:8083/
↪geoserver/wms?service=WMS&vers
↪1.0&request=GetMap&layers=geos
↪styles=&bbox=-130.85168,20.70
↪0054,54.1141&width=768&height=
↪format=application/openlayers
```

Finally, you can add a ColorMap on the selected channel as the following:

```
<RasterSymbolizer>
 <Opacity>1.0</Opacity>
 <ChannelSelection>
   <GrayChannel>
     <SourceChannelName>
            <ogc:Function name="
            <ogc:Literal>B1</ogc
            <ogc:Literal>7</ogc
         </ogc:Function>
     </SourceChannelName>
   </GrayChannel>
 </ChannelSelection>
 <ColorMap>
↪  <ColorMapEntry color="#0000
↪ <ColorMapEntry color="#009933
↪<ColorMapEntry color="#ff9900
↪ <ColorMapEntry color="#ff000
 </ColorMap>
</RasterSymbolizer>
```



ContrastEnhancement

The `<ContrastEnhancement>` element is used to adjust the relative brightness of the image data. A `<ContrastEnhancement>` element can be specified for the entire image, or in individual `Channel` elements. In this way, different enhancements can be used on each channel.

There are three types of enhancements possible:

- Normalize
- Histogram
- GammaValue

`<Normalize>` means to expand the contrast so that the minimum quantity is mapped to minimum brightness, and the maximum quantity is mapped to maximum brightness.

<Histogram> is similar to Normalize, but the algorithm used attempts to produce an image with an equal number of pixels at all brightness levels.

<GammaValue> is a scaling factor that adjusts the brightness of the image. A value less than one (1) darkens the image, and a value greater than one (1) brightens it. The default is 1 (no change).

These examples turn on Normalize and Histogram, respectively:

```
<ContrastEnhancement>
        <Normalize/>
</ContrastEnhancement>
```

```
<ContrastEnhancement>
        <Histogram/>
</ContrastEnhancement>
```

This example increases the brightness of the image by a factor of two.

```
<ContrastEnhancement>
        <GammaValue>2</GammaValue>
</ContrastEnhancement>
```

It is also possible to customize Normalize Contrast Enhancement element for the RasterSymbolizer. 3 new VendorOptions are supported:

- <VendorOption name="algorithm">ALGORITHM_NAME</VendorOption> to control the algorithm to apply
- <VendorOption name="minValue">MIN_VALUE</VendorOption> to control the min value for the algorithm
- <VendorOption name="maxValue">MAX_VALUE</VendorOption> to control the max value for the algorithm

Supported algorithms are:

- **StretchToMinimumMaximum** it will linearly stretch the source raster by linearly mapping values within the [MIN_VALUE, MAX_VALUE] range to [0,255]. This will also automatically result into a clip of the values outside the specified input range.
- **ClipToMinimumMaximum** it will result into a clamp operation. Values smaller than MIN_VALUE will be forced to MIN_VALUE. Values greater than MAX_VALUE will be forced to MAX_VALUE. Values in the [MIN_VALUE, MAX_VALUE] range will passthrough unchanged.
- **ClipToZero** is similar to ClipToMinimumMaximum. However, values outside the [MIN_VALUE, MAX_VALUE] range will be forced to be 0.

Note: The target data type for the stretch algorithm is **always** byte (this might change in the future). This means that if the MAX_VALUE for the Clip oriented algorithms is greater than 255 an implicit clamp will apply anyway to clamp to 255.

Here below some examples

```
<ContrastEnhancement>
  <Normalize>
   <VendorOption name=
↪"algorithm">StretchToMinimumMaximum</VendorOption>
   <VendorOption name="minValue">50</VendorOption>
   <VendorOption name="maxValue">100</VendorOption>
  </Normalize>
</ContrastEnhancement>
```

This example will apply a Normalized ContrastEnhancement by linearly stretch from pixel values [50, 100] to [0, 255]

```
<ContrastEnhancement>
  <Normalize>
   <VendorOption␣
↪name="algorithm">ClipToMinimumMaximum</VendorOption>
   <VendorOption name="minValue">50</VendorOption>
   <VendorOption name="maxValue">100</VendorOption>
  </Normalize>
</ContrastEnhancement>
```

```
<ContrastEnhancement>
  <Normalize>
   <VendorOption␣
↪name="algorithm">ClipToMinimumMaximum</VendorOption>
   <VendorOption name="minValue">50</VendorOption>
   <VendorOption name="maxValue">100</VendorOption>
  </Normalize>
</ContrastEnhancement>
```

Here below a more complex example that shows the possibility to control the values from a client using env functions. This is extremely interesting for interactive applications.

```
...
<ContrastEnhancement>
        <Normalize>
         <VendorOption name="algorithm">
           <ogc:Function name="env">
                 <ogc:Literal>algorithm</ogc:Literal>
                 ␣
↪    <ogc:Literal>StretchToMinimumMaximum</ogc:Literal>
           </ogc:Function>
         </VendorOption>
         <VendorOption name='minValue'>
           <ogc:Function name="env">
                 <ogc:Literal>minValue</ogc:Literal>
                 <ogc:Literal>10</ogc:Literal>
           </ogc:Function>
         </VendorOption>
         <VendorOption name='maxValue'>
           <ogc:Function name="env">
                 <ogc:Literal>maxValue</ogc:Literal>
                 <ogc:Literal>1200</ogc:Literal>
           </ogc:Function>
         </VendorOption>
        </Normalize>
```

```
                    </ContrastEnhancement>
                    ...
```

ShadedRelief

Warning: Support for this element has not been implemented yet.

The `<ShadedRelief>` element can be used to create a 3-D effect, by selectively adjusting brightness. This is a nice effect to use on an elevation dataset. There are two types of shaded relief possible.

- BrightnessOnly
- ReliefFactor

BrightnessOnly, which takes no parameters, applies shading in WHAT WAY? ReliefFactor sets the amount of exaggeration of the shading (for example, to make hills appear higher). According to the OGC SLD specification, a value of around 55 gives "reasonable results" for Earth-based datasets:

```
<ShadedRelief>
        <BrightnessOnly />
        <ReliefFactor>55</ReliefFactor>
</ShadedRelief>
```

The above example turns on Relief shading in WHAT WAY?

OverlapBehavior

Warning: Support for this element has not been implemented yet.

Sometimes raster data is comprised of multiple image sets. Take, for example, a satellite view of the Earth at night . As all of the Earth can't be in nighttime at once, a composite of multiple images are taken. These images are georeferenced, and pieced together to make the finished product. That said, it is possible that two images from the same dataset could overlap slightly, and the OverlapBehavior element is designed to determine how this is handled. There are four types of OverlapBehavior:

- AVERAGE
- RANDOM
- LATEST_ON_TOP
- EARLIEST_ON_TOP

AVERAGE takes each overlapping point and displays their average value. **RANDOM** determines which image gets displayed according to chance (which can sometimes result in a crisper image).

LATEST_ON_TOP and **EARLIEST_ON_TOP** sets the determining factor to be the internal timestamp on each image in the dataset. None of these elements have any parameters, and are all called in the same way:

```
<OverlapBehavior>
        <AVERAGE />
</OverlapBehavior>
```

The above sets the OverlapBehavior to AVERAGE.

ImageOutline

Warning: Support for this element has not been implemented yet.

Given the situation mentioned previously of the image composite, it is possible to style each image so as to have an outline. One can even set a fill color and opacity of each image; a reason to do this would be to "gray-out" an image. To use ImageOutline, you would define a <LineSymbolizer> or <PolygonSymbolizer> inside of the element:

```
<ImageOutline>
        <LineSymbolizer>
        <Stroke>

↪    <CssParameter name="stroke">#0000ff</CssParameter>
        </Stroke>
        </LineSymbolizer>
</ImageOutline>
```

The above would create a border line (colored blue with a one pixel default thickness) around each image in the dataset.

6.2.5 SLD Extensions in GeoServer

GeoServer provides a number of vendor-specific extensions to SLD 1.0. Although not portable to other applications, these extensions make styling more powerful and concise and allow for the generation of better-looking maps.

Geometry transformations in SLD

SLD symbolizers may contain an optional `<Geometry>` element, which allows specifying which geometry attribute is to be rendered. In the common case of a featuretype with a single geometry attribute this element is usually omitted, but it is useful when a featuretype has multiple geometry-valued attributes.

SLD 1.0 requires the `<Geometry>` content to be a `<ogc:PropertyName>`. GeoServer extends this to allow a general SLD expression to be used. The expression can contain

filter functions that manipulate geometries by transforming them into something different. This facility is called SLD *geometry transformations*.

GeoServer provides a number of filter functions that can transform geometry. A full list is available in the *Filter Function Reference*. They can be used to do things such as extracting line vertices or endpoints, offsetting polygons, or buffering geometries.

Geometry transformations are computed in the geometry's original coordinate reference system, before any reprojection and rescaling to the output map is performed. For this reason, transformation parameters must be expressed in the units of the geometry CRS. This must be taken into account when using geometry transformations at different screen scales, since the parameters will not change with scale.

Examples

Let's look at some examples.

Extracting vertices

Here is an example that allows one to extract all the vertices of a geometry, and make them visible in a map, using the `vertices` function:

```
<PointSymbolizer>
  <Geometry>
    <ogc:Function name="vertices">
       <ogc:PropertyName>the_geom</ogc:PropertyName>
    </ogc:Function>
  </Geometry>
  <Graphic>
    <Mark>
      <WellKnownName>square</WellKnownName>
      <Fill>
        <CssParameter name="fill">#FF0000</CssParameter>
      </Fill>
    </Mark>
    <Size>6</Size>
  </Graphic>
</PointSymbolizer>
```

View the full "Vertices" SLD

Applied to the sample *tasmania_roads* layer this will result in:

Start and end point

The *startPoint* and *endPoint* functions can be used to extract the start and end point of a line.

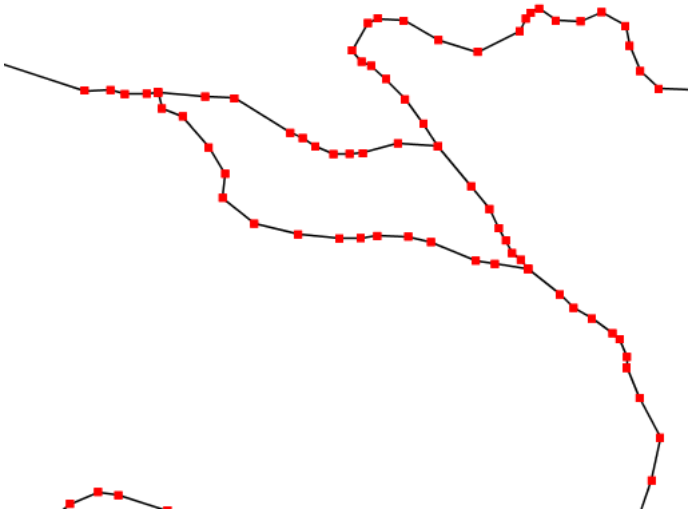


Fig. 6.78: *Extracting and showing the vertices out of a geometry*

```
                        <PointSymbolizer>
                          <Geometry>
                            <ogc:Function name="startPoint">
                              <ogc:PropertyName>the_geom</ogc:PropertyName>
                            </ogc:Function>
                          </Geometry>
                          <Graphic>
                            <Mark>
                              <WellKnownName>square</WellKnownName>
                              <Stroke>
                                <CssParameter name="stroke">0x00FF00</CssParameter>
                                <CssParameter name="stroke-width">1.5</CssParameter>
                              </Stroke>
                            </Mark>
                            <Size>8</Size>
                          </Graphic>
                        </PointSymbolizer>
                        <PointSymbolizer>
                          <Geometry>
                            <ogc:Function name="endPoint">
                              <ogc:PropertyName>the_geom</ogc:PropertyName>
                            </ogc:Function>
                          </Geometry>
                          <Graphic>
                            <Mark>
                              <WellKnownName>circle</WellKnownName>
                              <Fill>
                                <CssParameter name="fill">0xFF0000</CssParameter>
                              </Fill>
                            </Mark>
                            <Size>4</Size>
```

```
      </Graphic>
    </PointSymbolizer>
```

View the full "StartEnd" SLD

Applied to the sample *tasmania_roads* layer this will result in:

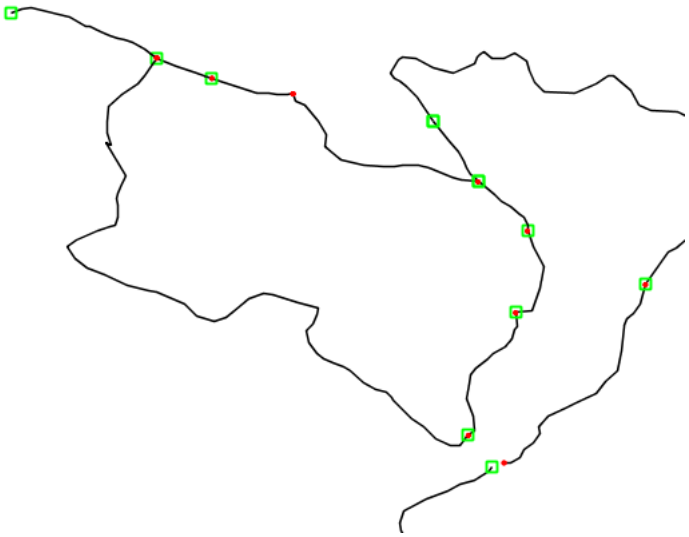


Fig. 6.79: *Extracting start and end point of a line*

Drop shadow

The *offset* function can be used to create drop shadow effects below polygons. Notice that the offset values reflect the fact that the data used in the example is in a geographic coordinate system.

```
    <PolygonSymbolizer>
      <Geometry>
        <ogc:Function name="offset">
          <ogc:PropertyName>the_geom</ogc:PropertyName>
          <ogc:Literal>0.00004</ogc:Literal>
          <ogc:Literal>-0.00004</ogc:Literal>
        </ogc:Function>
      </Geometry>
      <Fill>
        <CssParameter name="fill">#555555</CssParameter>
      </Fill>
    </PolygonSymbolizer>
```

View the full "Shadow" SLD

Applied to the sample *tasmania_roads* layer this will result in:

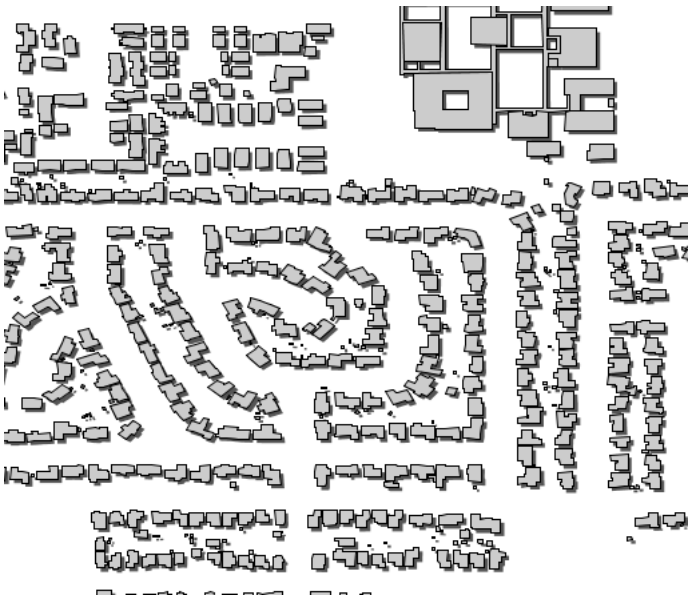


Fig. 6.80: *Dropping building shadows*

Performance tips

GeoServer's filter functions contain a number of set-related or constructive geometric functions, such as `buffer`, `intersection`, `difference` and others. These can be used as geometry transformations, but they be can quite heavy in terms of CPU consumption so it is advisable to use them with care. One strategy is to activate them only at higher zoom levels, so that fewer features are processed.

Buffering can often be visually approximated by using very large strokes together with round line joins and line caps. This avoids incurring the performance cost of a true geometric buffer transformation.

Adding new transformations

Additional filter functions can be developed in Java and then deployed in a JAR file as a GeoServer plugin. A guide is not available at this time, but see the GeoTools `main` module for examples.

Rendering Transformations

Rendering Transformations allow processing to be carried out on datasets within the GeoServer rendering pipeline. A typical transformation computes a derived or aggregated result from the input data, allowing various useful visualization effects to be obtained. Transformations may transform data from one format into another (i.e vector to raster or vice-versa), to provide an appropriate format for display.

The following table lists examples of various kinds of rendering transformations available in GeoServer:

Type	Examples
Raster-to-Vector	**Contour** extracts contours from a DEM raster. **RasterAsPointCollections** extracts a vector field from a multi-band raster
Vector-to-Raster	**BarnesSurfaceInterpolation** computes a surface from scattered data points. **Heatmap** computes a heatmap surface from weighted data points.
Vector-to-Vector	**PointStacker** aggregates dense point data into clusters.

Rendering transformations are invoked within SLD styles. Parameters may be supplied to control the appearance of the output. The rendered output for the layer is produced by applying the styling rules and symbolizers in the SLD to the result of transformation.

Rendering transformations are implemented using the same mechanism as *Process Cookbook*. They can thus also be executed via the WPS protocol, if required. Conversely, any WPS process can be executed as a transformation, as long as the input and output are appropriate for use within an SLD.

This section is a general guide to rendering transformation usage in GeoServer. For details of input, parameters, and output for any particular rendering transformation, refer to its own documentation.

Installation

Using Rendering Transformations requires the WPS extension to be installed. See *Installing the WPS extension*.

Note: The WPS service does not need to be **enabled** to use Rendering Transformations. To avoid unwanted consumption of server resources it may be desirable to disable the WPS service if it is not being used directly.

Usage

Rendering Transformations are invoked by adding the `<Transformation>` element to a `<FeatureTypeStyle>` element in an SLD document. This element specifies the name of the transformation process, and usually includes parameter values controlling the operation of the transformation.

The `<Transformation>` element syntax leverages the OGC Filter function syntax. The content of the element is a `<ogc:Function>` with the name of the rendering transformation process. Transformation processes may accept some number of parameters, which may be either required (in which case they must be specified), or optional (in which case they may be omitted if the default value is acceptable). Parameters are supplied as name/value pairs. Each parameter's name and value are supplied via another function `<ogc:Function name="parameter">`. The first argument to

this function is an `<ogc:Literal>` containing the name of the parameter. The optional following arguments provide the value for the parameter (if any). Some parameters accept only a single value, while others may accept a list of values. As with any filter function argument, values may be supplied in several ways:

- As a literal value
- As a computed expression
- As an SLD environment variable, whose actual value is supplied in the WMS request (see *Variable substitution in SLD*).
- As a predefined SLD environment variable (which allows obtaining values for the current request such as output image width and height).

The order of the supplied parameters is not significant.

Most rendering transformations take as input a dataset to be transformed. This is supplied via a special named parameter which does not have a value specified. The name of the parameter is determined by the particular transformation being used. When the transformation is executed, the input dataset is passed to it via this parameter.

The input dataset is determined by the same query mechanism as used for all WMS requests, and can thus be filtered in the request if required.

In rendering transformations which take as input a featuretype (vector dataset) and convert it to a raster dataset, in order to pass validation the SLD needs to mention the geometry attribute of the input dataset (even though it is not used). This is done by specifying the attribute name in the symbolizer `<Geometry>` element.

The output of the rendering transformation is styled using symbolizers appropriate to its format: *PointSymbolizer*, *LineSymbolizer*, *PolygonSymbolizer*, and *TextSymbolizer* for vector data, and *RasterSymbolizer* for raster coverage data.

If it is desired to display the input dataset in its orginal form, or transformed in another way, there are two options:

- Another `<FeatureTypeStyle>` can be used in the same SLD
- Another SLD can be created, and the layer displayed twice using the different SLDs

Notes

- Rendering transformations may not work correctly in tiled mode, unless they have been specifically written to accomodate it.

Examples

Contour extraction

`ras:Contour` is a **Raster-to-Vector** rendering transformation which extracts contour lines at specified lev-

els from a raster DEM. The following SLD invokes the transformation and styles the contours as black lines.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<StyledLayerDescriptor version="1.0.0"
  xsi:schemaLocation="http://www.opengis.net/sld StyledLayerDescriptor.xsd"
  xmlns="http://www.opengis.net/sld"
  xmlns:ogc="http://www.opengis.net/ogc"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <NamedLayer>
    <Name>contour_dem</Name>
    <UserStyle>
      <Title>Contour DEM</Title>
      <Abstract>Extracts contours from DEM</Abstract>
      <FeatureTypeStyle>
        <Transformation>
          <ogc:Function name="ras:Contour">
            <ogc:Function name="parameter">
              <ogc:Literal>data</ogc:Literal>
            </ogc:Function>
            <ogc:Function name="parameter">
              <ogc:Literal>levels</ogc:Literal>
              <ogc:Literal>1100</ogc:Literal>
              <ogc:Literal>1200</ogc:Literal>
              <ogc:Literal>1300</ogc:Literal>
              <ogc:Literal>1400</ogc:Literal>
              <ogc:Literal>1500</ogc:Literal>
              <ogc:Literal>1600</ogc:Literal>
              <ogc:Literal>1700</ogc:Literal>
              <ogc:Literal>1800</ogc:Literal>
            </ogc:Function>
          </ogc:Function>
        </Transformation>
        <Rule>
          <Name>rule1</Name>
          <Title>Contour Line</Title>
          <LineSymbolizer>
            <Stroke>
              <CssParameter name="stroke">#000000</CssParameter>
              <CssParameter name="stroke-width">1</CssParameter>
            </Stroke>
          </LineSymbolizer>
          <TextSymbolizer>
            <Label>
              <ogc:PropertyName>value</ogc:PropertyName>
            </Label>
            <Font>
              <CssParameter name="font-family">Arial</CssParameter>
              <CssParameter name="font-style">Normal</CssParameter>
              <CssParameter name="font-size">10</CssParameter>
```

```
                    </Font>
                    <LabelPlacement>
                      <LinePlacement/>
                    </LabelPlacement>
                    <Halo>
                      <Radius>
                        <ogc:Literal>2</ogc:Literal>
                      </Radius>
                      <Fill>
      <CssParameter name="fill">#FFFFFF</CssParameter>
    <CssParameter name="fill-opacity">0.6</CssParameter>
                      </Fill>
                    </Halo>
                    <Fill>
      <CssParameter name="fill">#000000</CssParameter>
                    </Fill>
                    <Priority>2000</Priority>
    <VendorOption name="followLine">true</VendorOption>
        <VendorOption name="repeat">100</VendorOption>
                    <VendorOption name="maxDisplacement">50</VendorOption>
    <VendorOption name="maxAngleDelta">30</VendorOption>
                  </TextSymbolizer>
                </Rule>
              </FeatureTypeStyle>
            </UserStyle>
          </NamedLayer>
        </StyledLayerDescriptor>
```

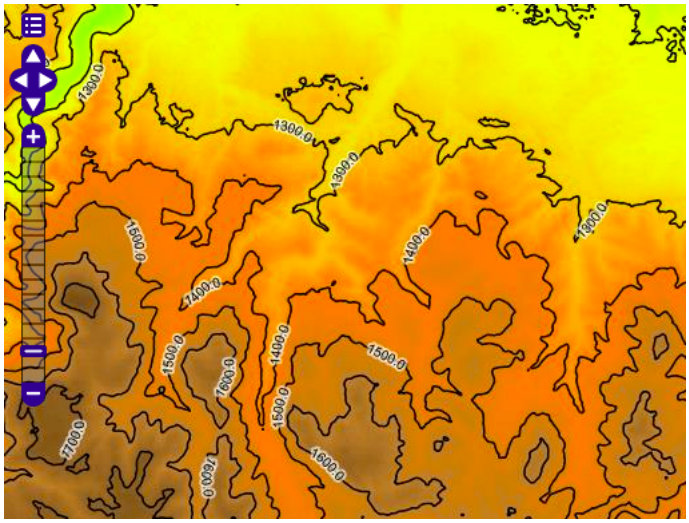
Key aspects of the SLD are:

- **Lines 14-15** define the rendering transformation, using the process `ras:Contour`.
- **Lines 16-18** supply the input data parameter, named `data` in this process.
- **Lines 19-29** supply values for the process's `levels` parameter, which specifies the elevation levels for the contours to extract.
- **Lines 35-40** specify a `LineSymbolizer` to style the contour lines.
- **Lines 41-70** specify a `TextSymbolizer` to show the contour levels along the lines.

The result of using this transformation is shown in the following map image (which also shows the underlying DEM raster):

Heatmap generation

`vec:Heatmap` is a **Vector-to-Raster** rendering transformation which generates a heatmap surface from weighted point data. The following SLD invokes a Heatmap rendering transformation on a featuretype



with point geometries and an attribute `pop2000` supplying the weight for the points (in this example, a dataset of world urban areas is used). The output is styled using a color ramp across the output data value range [0 .. 1].

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<StyledLayerDescriptor version="1.0.0"
    xsi:schemaLocation="http://www.opengis.net/sld StyledLayerDescriptor.xsd"
    xmlns="http://www.opengis.net/sld"
    xmlns:ogc="http://www.opengis.net/ogc"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <NamedLayer>
    <Name>Heatmap</Name>
    <UserStyle>
      <Title>Heatmap</Title>
      <Abstract>A heatmap surface showing population density</Abstract>
      <FeatureTypeStyle>
        <Transformation>
          <ogc:Function name="vec:Heatmap">
            <ogc:Function name="parameter">
              <ogc:Literal>data</ogc:Literal>
            </ogc:Function>
            <ogc:Function name="parameter">
              <ogc:Literal>weightAttr</ogc:Literal>
              <ogc:Literal>pop2000</ogc:Literal>
            </ogc:Function>
            <ogc:Function name="parameter">
              <ogc:Literal>radiusPixels</ogc:Literal>
              <ogc:Function name="env">
                <ogc:Literal>radius</ogc:Literal>
                <ogc:Literal>100</ogc:Literal>
              </ogc:Function>
            </ogc:Function>
            <ogc:Function name="parameter">
              <ogc:Literal>pixelsPerCell</ogc:Literal>
```

```
32                                  <ogc:Literal>10</ogc:Literal>
33                                </ogc:Function>
34                                <ogc:Function name="parameter">
35                                  <ogc:Literal>outputBBOX</ogc:Literal>
36                                  <ogc:Function name="env">
37                                    <ogc:Literal>wms_bbox</ogc:Literal>
38                                  </ogc:Function>
39                                </ogc:Function>
40                                <ogc:Function name="parameter">
41                                  <ogc:Literal>outputWidth</ogc:Literal>
42                                  <ogc:Function name="env">
43                                    <ogc:Literal>wms_width</ogc:Literal>
44                                  </ogc:Function>
45                                </ogc:Function>
46                                <ogc:Function name="parameter">
47                                  <ogc:Literal>outputHeight</ogc:Literal>
48                                  <ogc:Function name="env">
49                                    <ogc:Literal>wms_height</ogc:Literal>
50                                  </ogc:Function>
51                                </ogc:Function>
52                              </ogc:Function>
53                            </Transformation>
54                            <Rule>
55                              <RasterSymbolizer>
56                              <!-- specify geometry attribute to pass validation -->
57                                <Geometry>
58                                  <ogc:PropertyName>the_geom</ogc:PropertyName></Geometry>
59                                <Opacity>0.6</Opacity>
60                                <ColorMap type="ramp" >
61                                  <ColorMapEntry color="#FFFFFF" quantity="0" label="nodata"
62                                    opacity="0"/>
63                                  <ColorMapEntry color="#FFFFFF" quantity="0.02" label="nodata"
64                                    opacity="0"/>
65                                  <ColorMapEntry color="#4444FF" quantity=".1" label="nodata"/>
66                                  <ColorMapEntry color="#FF0000" quantity=".5" label="values" />
67                                  <ColorMapEntry color="#FFFF00" quantity="1.0" label="values" />
68                                </ColorMap>
69                              </RasterSymbolizer>
70                            </Rule>
71                          </FeatureTypeStyle>
72                        </UserStyle>
73                      </NamedLayer>
74                    </StyledLayerDescriptor>
```

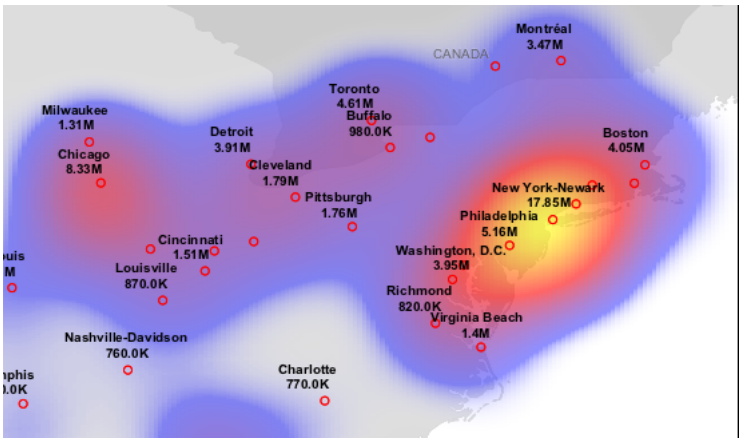
Key aspects of the SLD are:

- **Lines 14-15** define the rendering transformation, using the process `vec:Heatmap`.
- **Lines 16-18** supply the input data parameter, named `data` in this

process.

- **Lines 19-22** supply a value for the process's `weightAttr` parameter, which specifies the input attribute providing a weight for each data point.
- **Lines 23-29** supply the value for the `radiusPixels` parameter, which controls the "spread" of the heatmap around each point. In this SLD the value of this parameter may be supplied by a SLD substitution variable called `radius`, with a default value of `100` pixels.
- **Lines 30-33** supply the `pixelsPerCell` parameter, which controls the resolution at which the heatmap raster is computed.
- **Lines 34-38** supply the `outputBBOX` parameter, which is given the value of the standard SLD environment variable `wms_bbox`.
- **Lines 40-45** supply the `outputWidth` parameter, which is given the value of the standard SLD environment variable `wms_width`.
- **Lines 46-52** supply the `outputHeight` parameter, which is given the value of the standard SLD environment variable `wms_height`.
- **Lines 55-70** specify a `RasterSymbolizer` to style the computed raster surface. The symbolizer contains a ramped color map for the data range [0..1].
- **Line 58** specifies the geometry attribute of the input featuretype, which is necessary to pass SLD validation.

  This transformation styles a layer to produce a heatmap surface for the data in the requested map extent, as shown in the image below. (The map image also shows the original input data points styled by another SLD, as well as a base map layer.)



Running map algebra on the fly using Jiffle

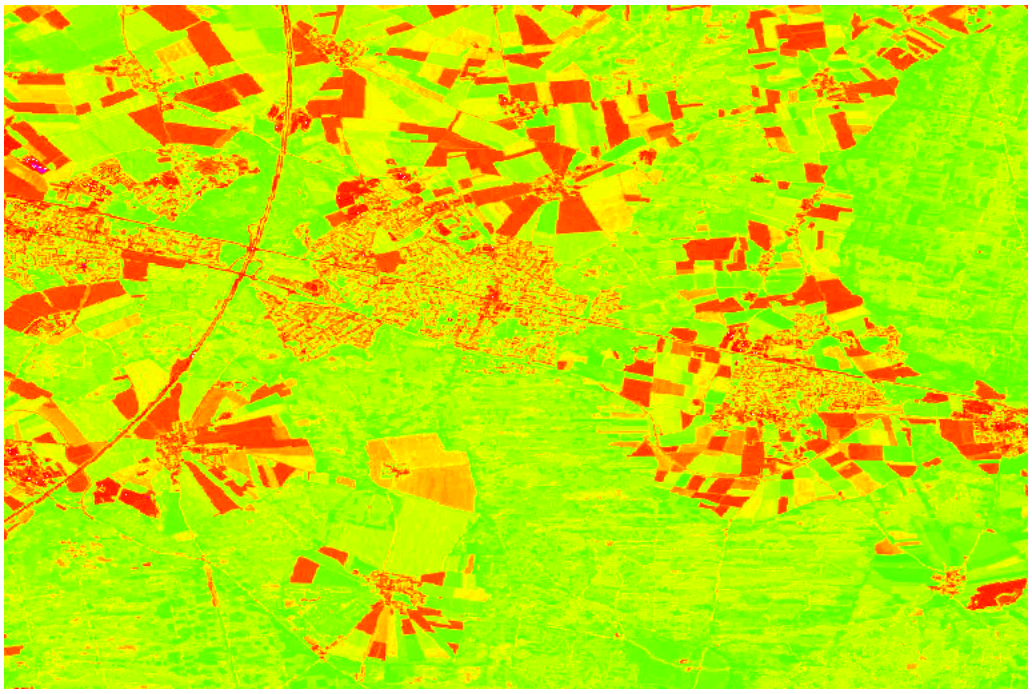
The `Jiffle` rendering transformation allows to run map algebra on the bands of an input raster layer using the Jiffle language. For example, the following style computes the NDVI index from a 13 bands Sentinel 2 image, in which the red and NIR bands are the

forth and eight bands (Jiffle band indexes are zero based), and then displays the resulting index with a color map:

```
<?xml version="1.0" encoding="UTF-8"?>
<StyledLayerDescriptor xmlns="http://www.opengis.net/sld" xmlns:ogc="http://www.opengis.net/ogc" xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.opengis.net/sld http://schemas.opengis.net/sld/1.0.0/StyledLayerDescriptor.xsd" version="1.0.0">
  <NamedLayer>
    <Name>Sentinel2 NDVI</Name>
    <UserStyle>
      <Title>NDVI</Title>
      <FeatureTypeStyle>
        <Transformation>
          <ogc:Function name="ras:Jiffle">
            <ogc:Function name="parameter">
              <ogc:Literal>coverage</ogc:Literal>
            </ogc:Function>
            <ogc:Function name="parameter">
              <ogc:Literal>script</ogc:Literal>
              <ogc:Literal>
                nir = src[7];
                vir = src[3];
                dest = (nir - vir) / (nir + vir);
              </ogc:Literal>
            </ogc:Function>
          </ogc:Function>
        </Transformation>
        <Rule>
          <RasterSymbolizer>
            <Opacity>1.0</Opacity>
            <ColorMap>
              <ColorMapEntry color="#000000" quantity="-1" />
              <ColorMapEntry color="#0000ff" quantity="-0.75" />
              <ColorMapEntry color="#ff00ff" quantity="-0.25" />
              <ColorMapEntry color="#ff0000" quantity="0" />
              <ColorMapEntry color="#ffff00" quantity="0.5" />
              <ColorMapEntry color="#00ff00" quantity="1" />
            </ColorMap>
          </RasterSymbolizer>
        </Rule>
      </FeatureTypeStyle>
    </UserStyle>
  </NamedLayer>
</StyledLayerDescriptor>
```

Here are a view of the area, using the visible color bands:

and then the display of the NDVI index computed with the above style:



Graphic symbology in GeoServer

Graphic symbology is supported via the SLD `<Graphic>` element. This element can appear in several contexts in SLD:

- in a *PointSymbolizer*, to display symbols at points
- in the `<Stroke>/<GraphicStroke>` element of a *LineSymbolizer* and *PolygonSymbolizer*, to display repeated symbols along lines and polygon boundaries.
- in the `<Stroke>/<GraphicFill>` element of a *LineSymbolizer* and *PolygonSymbolizer*, to fill lines and polygon boundaries with tiled symbols.
- in the `<Fill>/<GraphicFill>` element of a *PolygonSymbolizer*, to fill polygons with tiled symbols (stippling).
- in a *TextSymbolizer* to display a graphic behind or instead of text labels (this is a GeoServer extension).

`<Graphic>` contains either a `<Mark>` or an `<ExternalGraphic>` element. **Marks** are pure vector symbols whose geometry is predefined but with stroke and fill defined in the SLD itself. **External Graphics** are external files (such as PNG images or SVG graphics) that contain the shape and color information defining how to render a symbol.

In standard SLD the `<Mark>` and `<ExternalGraphic>` names are fixed strings. GeoServer extends this by providing *dynamic symbolizers*, which allow computing symbol names on a per-feature basis by embedding CQL expressions in them.

Marks

GeoServer supports the standard SLD `<Mark>` symbols, a user-expandable set of extended symbols, and also TrueType Font glyphs. The symbol names are specified in the `<WellKnownName>` element.

See also the *PointSymbolizer* reference for further details, as well as the examples in the *Points* Cookbook section.

Standard symbols

The SLD specification mandates the support of the following symbols:

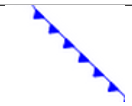
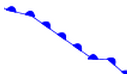
Name	Description
`square`	A square
`circle`	A circle
`triangle`	A triangle pointing up
`star`	five-pointed star
`cross`	A square cross with space around (not suitable for hatch fills)
`x`	A square X with space around (not suitable for hatch fills)

Shape symbols

The shape symbols set adds extra symbols that are not part of the basic set. Their names are prefixed by `shape://`

Name	Description
`shape://vertline`	A vertical line (suitable for hatch fills or to make railroad symbols)
`shape://horline`	A horizontal line (suitable for hatch fills)
`shape://slash`	A diagonal line leaning forwards like the "slash" keyboard symbol (suitable for diagonal hatches)
`shape://backslash`	Same as `shape://slash`, but oriented in the opposite direction
`shape://dot`	A very small circle with space around
`shape://plus`	A + symbol, without space around (suitable for cross-hatch fills)
`shape://times`	A "X" symbol, without space around (suitable for cross-hatch fills)
`shape://oarrow`	An open arrow symbol (triangle without one side, suitable for placing arrows at the end of lines)
`shape://carrow`	A closed arrow symbol (closed triangle, suitable for placing arrows at the end of lines)

The weather symbols are prefixed by the `extshape://` protocol in the SLD:

Name	Description	Produces
`extshape://triangle`	cold front	
`extshape://emicircle`	warm front	
`extshape://triangleemicircle`	stationary front	

You can use `extshape://` for a few additional built-in shapes:

`extshape://narrow`	North Arrow
`extshape://sarrow`	South Arrow

More complex symbols like Wind Barbs can be created with the `windbarbs://` prefix. There are some examples:

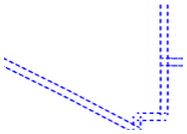
Name	Description
`windbarbs://default(15)[kts]`	*15* wind intensity with *[kts]* unit of measure
`windbarbs://default(9)[m/s]?hemisphere=s`	*9* wind intensity with *[m/s]* unit of measure, in the south hemisphere

Custom WKT Shapes

Custom shapes can be defined using your own Geometry. Geometry is defined using the same well-known-text format used for CQL_FILTER.

```
<LineSymbolizer>
  <Stroke>
    <GraphicStroke>
      <Graphic>
        <Mark>
          <WellKnownName>
↪wkt://MULTILINESTRING((-0.25 -0.25, -0.
↪125 -0.25), (0.125 -0.25, 0.25 -0.25), (-0.25 0.25, -
↪0.125 0.25), (0.125 0.25, 0.25 0.25))</WellKnownName>
          <Fill>

↪       <CssParameter name="fill">#0000ff</CssParameter>
          </Fill>
          <Stroke>

↪     <CssParameter name="stroke">#0000ff</CssParameter>

↪     <CssParameter name="stroke-width">1</CssParameter>
          </Stroke>
        </Mark>
        <Size>6</Size>
      </Graphic>
    </GraphicStroke>
  </Stroke>
</LineSymbolizer>
```

Which produces double dashed line:



You can also make use of curves when defining WKT:

```
<LineSymbolizer>
  <Stroke>
    <GraphicStroke>
      <Graphic>
        <Mark>
          <WellKnownName>wkt:/
↪/COMPOUNDCURVE((0 0, 0.25 0), CIRCULARSTRING(0.25
↪0, 0.5 0.5, 0.75 0), (0.75 0, 1 0))</WellKnownName>
          <Fill>

↪       <CssParameter name="fill">#0000ff</CssParameter>
```

```
            </Fill>
            <Stroke>
              <CssParameter name="stroke">#0000ff</CssParameter>
              <CssParameter name="stroke-width">1</CssParameter>
            </Stroke>
          </Mark>
          <Size>10</Size>
        </Graphic>
      </GraphicStroke>
    </Stroke>
  </LineSymbolizer>
```

Producing an "emi circle" line:

Bulk TTF marks

It is possible to create a mark using glyphs from any decorative or symbolic True Type Font, such as Wingdings, WebDings, or the many symbol fonts available on the internet. The syntax for specifying this is:

```
ttf://<fontname>#<hexcode>
```

where `fontname` is the full name of a TTF font available to GeoServer, and `hexcode` is the hexadecimal code of the symbol. To get the hex code of a symbol, use the "Char Map" utility available in most operating systems (Windows and Linux Gnome both have one).

For example, to use the "shield" symbol contained in the WebDings font, the Gnome `charmap` reports the symbol hex code as shown:

The SLD to use the shield glyph as a symbol is:

```
 <PointSymbolizer>
     <Graphic>
       <Mark>
         <WellKnownName>ttf://Webdings#0x0064</WellKnownName>
         <Fill>
           <CssParameter name="fill">#AAAAAA</CssParameter>
         </Fill>
         <Stroke/>
       </Mark>
     <Size>16</Size>
   </Graphic>
 </PointSymbolizer>
```

This results in the following map display:

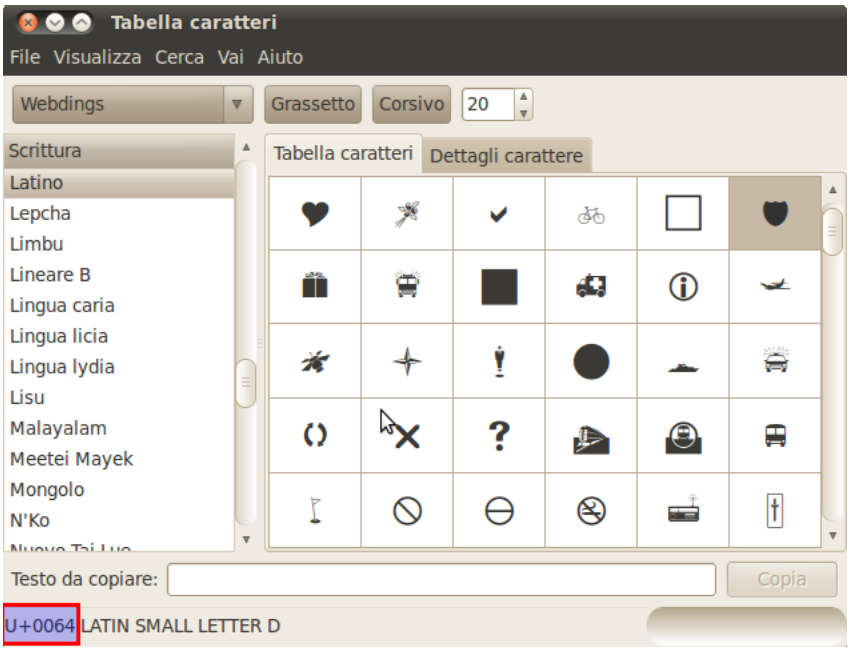


Fig. 6.81: *Selecting a symbol hex code in the Gnome char map*

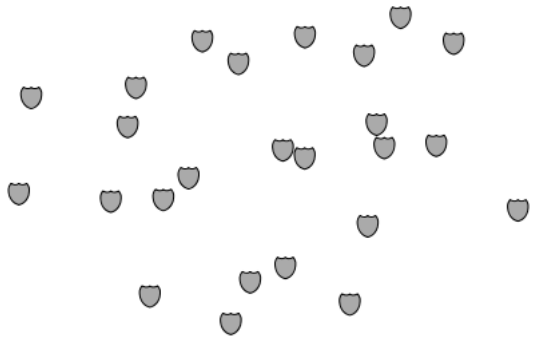


Fig. 6.82: *Shield symbols rendered on the map*

Extending the Mark subsytem using Java

The Mark subsystem is user-extensible. To do this using Java code, implement the `MarkFactory` interface and declare the implementation in the `META-INF/services/org.geotools.renderer.style.MarkFactory` file.

For further information see the Javadoc of the GeoTools MarkFactory, along with the following example code:

- The factory SPI registration file
- The TTFMarkFactory implementation
- The ShapeMarkFactory implementation

External Graphics

`<ExternalGraphic>` is the other way to define point symbology. Unlike marks, external graphics are used as-is, so the specification is somewhat simpler. The element content specifies a graphic `<OnlineResource>` using a URL or file path, and the graphic `<Format>` using a MIME type:

```
<PointSymbolizer>
    <Graphic>
       <ExternalGraphic>
          <OnlineResource xlink:type="simple"␣
↪xlink:href="http://mywebsite.com/pointsymbol.png" />
          <Format>image/png</Format>
       </ExternalGraphic>
    </Graphic>
</PointSymbolizer>
```

As with `<Mark>`, a `<Size>` element can be optionally specified. When using images as graphic symbols it is better to avoid resizing, as that may blur their appearance. Use images at their native resolution by omitting the `<Size>` element. In contrast, for SVG graphics specifying a `<Size>` is recommended. SVG files are a vector-based format describing both shape and color, so they scale cleanly to any size.

If the path of the symbol file is relative, the file is looked for under `$GEOSERVER_DATA_DIR/styles`. For example:

```
<PointSymbolizer>
  <Graphic>
    <ExternalGraphic>
      <OnlineResource␣
↪xlink:type="simple" xlink:href="burg02.svg" />
      <Format>image/svg+xml</Format>
    </ExternalGraphic>
    <Size>20</Size>
  </Graphic>
</PointSymbolizer>
```

In this example an SVG graphic is being used, so the size is specified explicitly.

Bulk WKT Shapes

It is possible to create a symbol set of your own custom marks using a property file.

Here is an `example.properties`:

```
zig=LINESTRING(0.
↪0 0.25, 0.25 0.25, 0.5 0.75, 0.75 0.25, 1.00 0.25)
block=POLYGON((0 0, 1 0, 1 1, 0 1, 0 0))
```

The SLD to use the symbols defined in `example.properties` is:

```
<PointSymbolizer>
  <Graphic>
    <ExternalGraphic>
      <OnlineResource
        xlink:type="simple"
        xlink:href="example.properties#zig" />
      <Format>wkt</Format>
    </ExternalGraphic>
    <Size>20</Size>
  </Graphic>
</PointSymbolizer>
```

Symbol Positioning

Graphic symbols are rendered so that the center of the graphic extent lies on the placement point (or points, in the case of repeated or tiled graphics). If it is desired to have a graphic offset from a point (such as a symbol which acts as a pointer) it is necessary to offset the visible portion of the graphic within the overall extent. For images this can be accomplished by extending the image with transparent pixels. For SVG graphics this can be done by surrounding the shape with an invisible rectangle with the desired relative position.

Dynamic symbolizers

In standard SLD, the `Mark/WellKnowName` element and the `ExternalGraphic/OnlineResource/@xlink:href` attribute are fixed strings. This means they have the same value for all rendered features. When the symbols to be displayed vary depending on feature attributes this restriction leads to very verbose styling, as a separate `Rule` and `Symbolizer` must be used for each different symbol.

GeoServer improves this by allowing *CQL expressions* to be embedded inside the content of both `WellKnownName` and `OnlineResource/@xlink:href`. When the names of the symbols can be derived from the feature attribute values, this provides much more compact styling. CQL expressions can be embedded in a `<WellKnownName>` content string or an `<OnlineResource>` `xlink:href` attribute by using the syntax:

```
${<cql expression>}
```

Note: Currently `xlink:href` strings must be valid URLs *before* expression expansion is performed. This means that the URL cannot be completely provided by an expression. The `xlink:href` string must explicitly include at least the prefix `http://`

The simplest form of expression is a single attribute name, such as `${STATE_ABBR}`. For example, suppose we want to display the flags of the US states using symbols whose file names match the state name. The following style specifies the flag symbols using a single rule:

```
<ExternalGraphic>
   <OnlineResource xlink:type="simple"
↪xlink:href="http://mysite.com/tn_${STATE_ABBR}.jpg"/>
   <Format>image/jpeg</Format>
</ExternalGraphic>
```

If manipulation of the attribute values is required a full CQL expression can be specified. For example, if the values in the `STATE_ABBR` attribute are uppercase but the URL requires a lowercase name, the CQL `strToLowerCase` function can be used:

```
<ExternalGraphic>
   <OnlineResource xlink:type="simple"
            xlink:href="http:/
↪/mysite.com/tn_${strToLowerCase(STATE_ABBR)}.jpg" />
   <Format>image/jpeg</Format>
</ExternalGraphic>
```

Variable substitution in SLD

Variable substitution in SLD is a GeoServer extension (starting in version 2.0.2) that allows passing values from WMS requests into SLD styles. This allows dynamically setting values such as colors, fonts, sizes and filter thresholds.

Variables are specified in WMS `GetMap` requests by using the `env` request parameter followed by a list of `name:value` pairs separated by semicolons:

```
...&env=name1:value1;name2=value2&...
```

In an SLD the variable values are accessed using the `env` function. The function retrieves a substitution variable value specified in the current request:

```
<ogc:Function name="env">
   <ogc:Literal>size</ogc:Literal>
</ogc:Function>
```

A default value can be provided. It will be used if the variable was not specified in the request:

```
<ogc:Function name="env">
   <ogc:Literal>size</ogc:Literal>
   <ogc:Literal>6</ogc:Literal>
</ogc:Function>
```

The `env` function can be used in an SLD anywhere an OGC expression is allowed. For example, it can be used in `CSSParameter` elements, in size and offset elements, and in rule filter expressions. It is also accepted in some places where full expressions are not allowed, such as in the `Mark/WellKnownName` element.

Predefined Variables

GeoServer has predefined variables which provide information about specific properties of the request output. These are useful when SLD parameters need to depend on output dimensions. The predefined variables are:

Name	Type	Description
`wms_bbox`	`ReferencedEnvelope`	the georeferenced extent of the request output
`wms_crs`	`CoordinateReferenceSystem`	the definition of the output coordinate reference system
`wms_srs`	`String`	the code for the output coordinate reference system
`wms_width`	`Integer`	the width (in pixels) of the output image
`wms_height`	`Integer`	the height (in pixels) of the output image
`wms_scale_denominator`	`Integer`	the denominator of the output map scale
`kmlOutputMode`	Either `vector` or empty	this variable gets set to `vector` when the kml generator is writing out vector features as placemarks, as opposed to ground overlays

Example

The following SLD symbolizer has been parameterized in three places, with default values provided in each case:

```
<PointSymbolizer>
  <Graphic>
    <Mark>
      <WellKnownName><ogc:Function name="env">
            <ogc:Literal>name</ogc:Literal>
            <ogc:Literal>square</ogc:Literal>
         </ogc:Function>
      </WellKnownName>
      <Fill>
        <CssParameter name="fill">
          #<ogc:Function name="env">
            <ogc:Literal>color</ogc:Literal>
            <ogc:Literal>FF0000</ogc:Literal>
          </ogc:Function>
        </CssParameter>
      </Fill>
    </Mark>
    <Size>
```

```
                                    <ogc:Function name="env">
                                       <ogc:Literal>size</ogc:Literal>
                                       <ogc:Literal>6</ogc:Literal>
                                    </ogc:Function>
                                 </Size>
                              </Graphic>
                           </PointSymbolizer>
```

Download the full SLD style

When no variables are provided in the WMS request, the SLD uses the default values and renders the sample `sf:bugsites` dataset as shown:

Fig. 6.83: *Default rendering*

If the request is changed to specify the following variable values:

```
&env=color:00FF00;name:triangle;size:12
```

the result is instead:

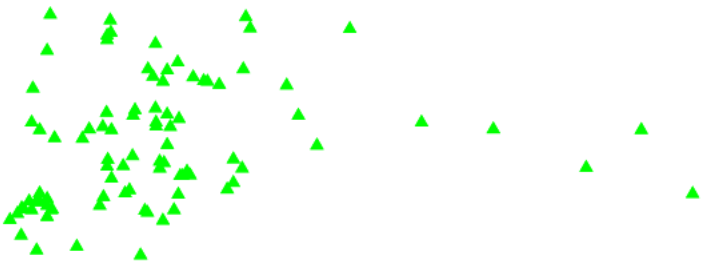


Fig. 6.84: *Rendering with variables supplied*

Specifying symbolizer sizes in ground units

The SLD 1.0 specification allows giving symbolizer sizes in a single unit of measure: pixels. This means that the size of symbolizers is the same at all zoom levels (which is usually the desired behaviour).

The Symbology Encoding 1.1 specification provides a `uom` attribute on `Symbolizer` elements. This allows specifying styling parameter sizes in ground units of metres or feet, as well as the default which is screen pixels. When ground units are used, the screen size of styled elements increases as the map is zoomed in to larger scales. GeoServer supports the SE 1.1 `uom` attribute in its extended SLD 1.0 support.

Note: This extended feature is officially supported in GeoServer 2.1.0. It is available in GeoServer 2.0.3 if the `-DenableDpiUomRescaling=true` system variable is specified for the JVM.

The value of the `uom` attribute is a URI indicating the desired unit. The units of measure supported are those given in the SE 1.1 specification:

```
http://www.opengeospatial.org/se/units/metre
http://www.opengeospatial.org/se/units/foot
http://www.opengeospatial.org/se/units/pixel
```

Note: The `px` override modifier for parameters values is not currently supported.

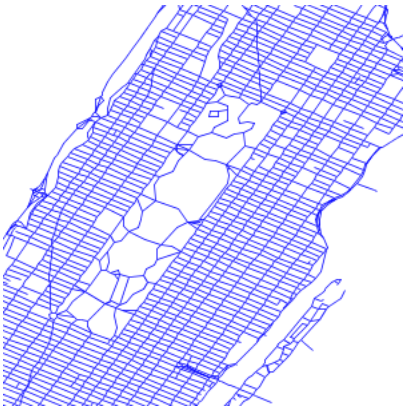
Example

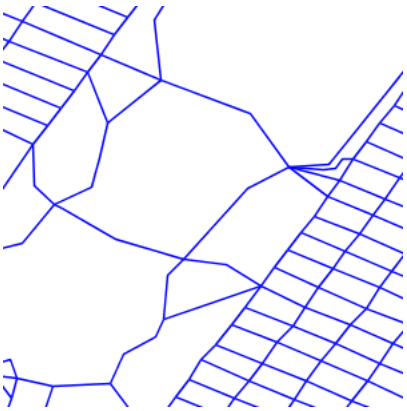
The following SLD shows the `uom` attribute used to specify the width of a `LineSymbolizer` in metres:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<StyledLayerDescriptor␣
↪version="1.0.0" xmlns="http://www.opengis.
↪net/sld" xmlns:ogc="http://www.opengis.net/ogc"
  xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xsi=
↪"http://www.w3.org/2001/XMLSchema-instance">
  <NamedLayer>
    <Name>5m blue line</Name>
    <UserStyle>
      <Title>tm blue line</Title>
      ␣
↪<Abstract>Default line style, 5m wide blue</Abstract>
      <FeatureTypeStyle>
        <Rule>
          <Title>Blue Line, 5m large</Title>
          <LineSymbolizer␣
↪uom="http://www.opengeospatial.org/se/units/metre">
            <Stroke>
              ␣
↪   <CssParameter name="stroke">#0000FF</CssParameter>
```

```
                                    ↪    <CssParameter name="stroke-width">5</CssParameter>
                                           </Stroke>
                                         </LineSymbolizer>
                                       </Rule>

                                      </FeatureTypeStyle>
                                    </UserStyle>
                                  </NamedLayer>
                                </StyledLayerDescriptor>
```

Applying the style to the `tiger:tiger_roads` dataset shows how the line widths increase as the map is zoomed in:

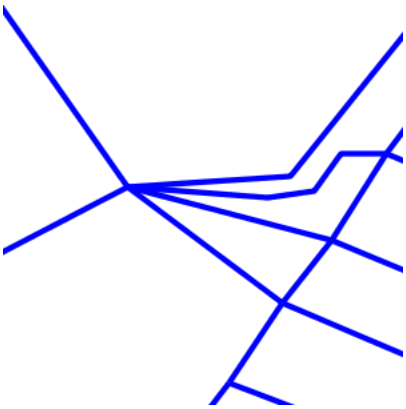




Label Obstacles

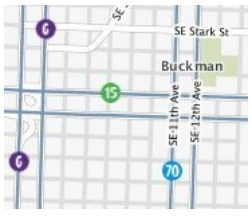
GeoServer implements an algorithm for label conflict resolution, to prevent labels from overlapping one another. By default this algorithm only considers conflicts with other labels. This can result in labels overlapping other symbolizers, which may produce an undesirable effect.

GeoServer supports a vendor option called `labelObstacle` that allows marking a symbolizer as an obstacle. This tells the labeller to avoid rendering labels that overlap it.

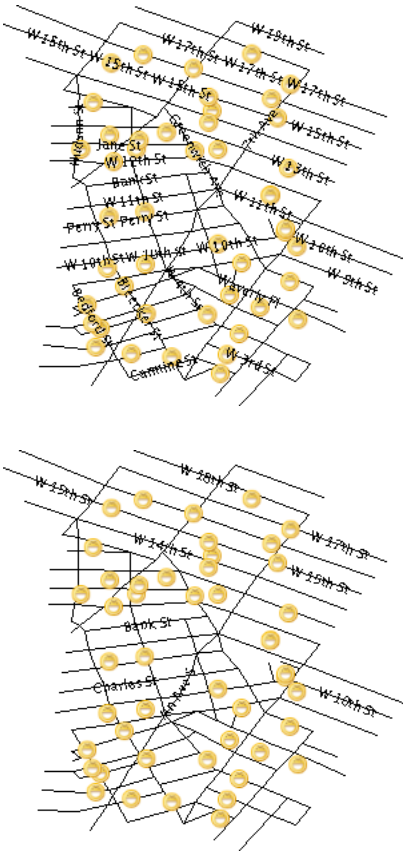


Warning: Beware of marking a line or poly symbolizer as a label obstacle. The label conflict resolving routine is based on the bounding box so marking as a label obstacle will result in no label overlapping not only the geometry itself, but its bounding box as well.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<StyledLayerDescriptor␣
↪version="1.0.0" xmlns="http://www.opengis.
↪net/sld" xmlns:ogc="http://www.opengis.net/ogc"
  xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xsi=
↪"http://www.w3.org/2001/XMLSchema-instance">
  <NamedLayer>
    <UserStyle>

  <FeatureTypeStyle>
    <Rule>
      <PointSymbolizer>
        <Graphic>
          <ExternalGraphic>
            <OnlineResource
              xlink:type="simple"
              xlink:href="smileyface.png" />
            <Format>image/png</Format>
          </ExternalGraphic>
          <Size>32</Size>
```



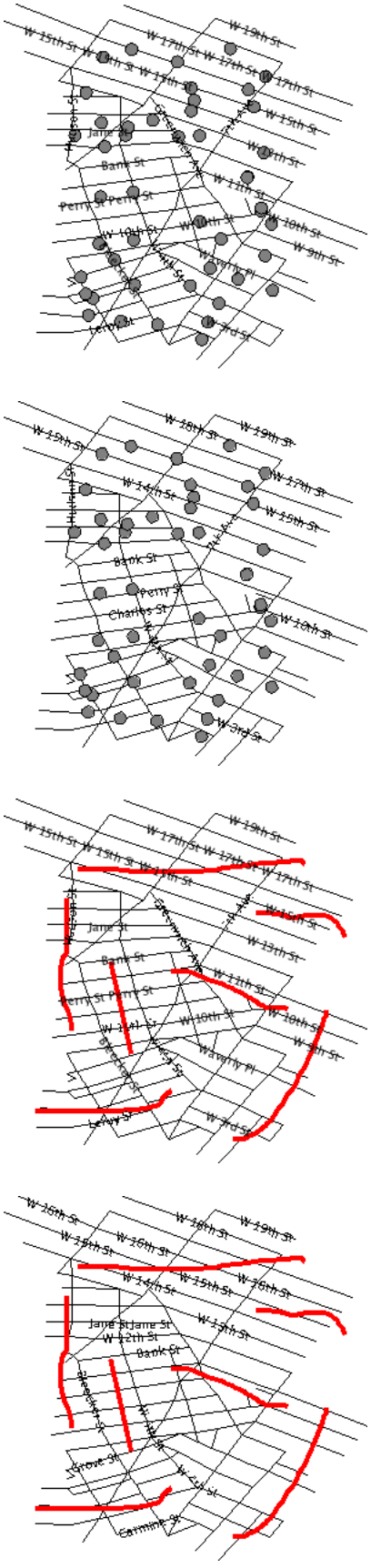
```
            </Graphic>
            <VendorOption␣
↪name="labelObstacle">true</VendorOption>
          </PointSymbolizer>
        </Rule>
      </FeatureTypeStyle>

        </UserStyle>
      </NamedLayer>
    </StyledLayerDescriptor>
```

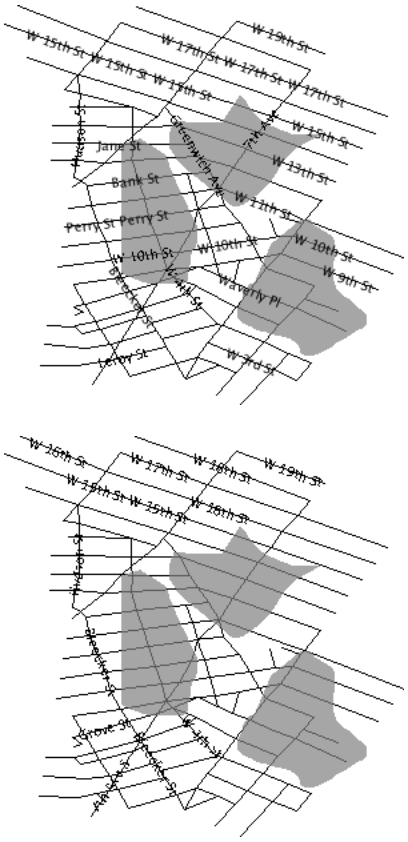


Applying the obstacle to a regular point style:

```
<PointSymbolizer>
  <Graphic>
    <ExternalGraphic>
      <OnlineResource
        xlink:type="simple"
        xlink:href="smileyface.png" />
      <Format>image/png</Format>
    </ExternalGraphic>
    <Size>32</Size>
  </Graphic>
  <VendorOption␣
↪name="labelObstacle">true</VendorOption>
</PointSymbolizer>
```

Applying the obstacle to line/polygon style style:


W 19th St
W 17th St
W 15th St
W 13th St
W 12th St
W 11th St
W 10th St
W 9th St
W 3rd St
Jane St
Bank St
Perry St
W 4th St
W 16th St
W 14th St
Grove St
Carmine St



Adding space around graphic fills

Starting with GeoServer 2.3.4 it is possible to add white space around symbols used inside graphic fills, effectively allowing to control the density of the symbols in the map.

```
<PolygonSymbolizer>
  <Fill>
    <GraphicFill>
      <Graphic>
        <ExternalGraphic>
          <OnlineResource xlink:type=
↪"simple" xlink:href="./rockFillSymbol.png"/>
          <Format>image/png</Format>
        </ExternalGraphic>
      </Graphic>
    </GraphicFill>
  </Fill>
  <VendorOption name="graphic-margin">10</VendorOption>
</PolygonSymbolizer>
```

The above forces 10 pixels of white space above, below and on either side of the symbol, effectively adding 20 pixels of white space between the symbols in the fill. The `graphic-margin` can be expressed, just like the CSS margin, in four different ways:

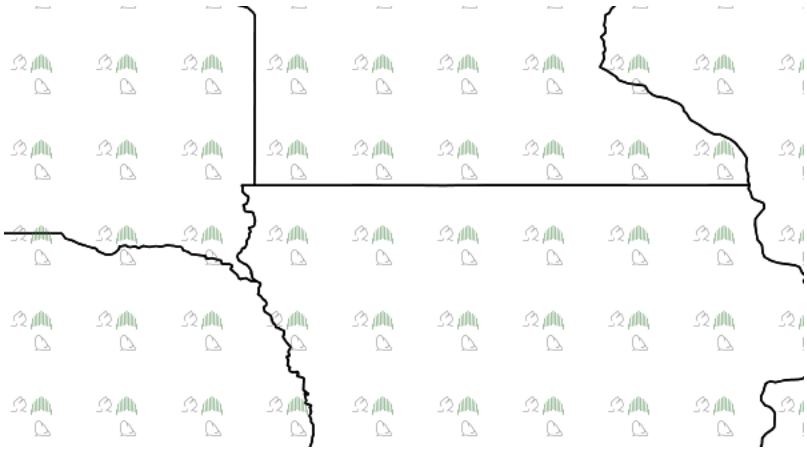
- top,right,bottom,left (one explicit value per margin)
- top,right-left,bottom (three values, with right and left sharing the

same value)

- top-bottom,right-left (two values, top and bottom sharing the same value)
- top-right-bottom-left (single value for all four margins)

The ability to specify different margins allows to use more than one symbol in a fill, and synchronize the relative positions of the various symbols to generate a composite fill:

```
<PolygonSymbolizer>
  <Fill>
    <GraphicFill>
      <Graphic>
        <ExternalGraphic>
          <OnlineResource xlink:type=
↪"simple" xlink:href="./boulderGeometry.png"/>
          <Format>image/png</Format>
        </ExternalGraphic>
      </Graphic>
    </GraphicFill>
  </Fill>
  <VendorOption␣
↪name="graphic-margin">35 17 17 35</VendorOption>
</PolygonSymbolizer>
<PolygonSymbolizer>
  <Fill>
    <GraphicFill>
      <Graphic>
        <ExternalGraphic>
          <OnlineResource xlink:type=
↪"simple" xlink:href="./roughGrassFillSymbol.png"/>
          <Format>image/png</Format>
        </ExternalGraphic>
      </Graphic>
    </GraphicFill>
  </Fill>
  <VendorOption␣
↪name="graphic-margin">16 16 32 32</VendorOption>
</PolygonSymbolizer>
```



Fills with randomized symbols

Starting with GeoServer 2.4.2 it is possible to generate fills by randomly repeating a symbol in the polygons to be filled. Or, to be more precise, generate the usual texture fill by repeating over and over a tile, whose contents is the random repetition of a fill. The random distribution is stable, so it will be the same across calls and tiles, and it's controlled by the seed used to generate the distribution.

The random fill is generated by specifying a GraphicFill with a Mark or ExternalGraphic, and then adding vendor options to control how the symbol is randomly repeated. Here is a table with options, default values, and possible values:

Option	Default value	Description
random	none	Activates random distribution of symbol. Possible values are **none**, **free**, **grid**. **none** disables random distribution, **free** generates a completely random distribution, **grid** will generate a regular grid of positions, and only randomizes the position of the symbol around the cell centers, providing a more even distribution in space
random-tile-size	256	Size the the texture fill tile that will contain the randomly distributed symbols
random-rotation	none	Activates random symbol rotation. Possible values are **none** (no rotation) or **free**
random-symbol-count	16	The number of symbols in the tile. The number of symbols actually painted can be lower, as the distribution will ensure no two symbols overlap with each other.
random-seed	0	The "seed" used to generate the random distribution. Changing this value will result in a different symbol distribution

Here is an example:

```
<sld:PolygonSymbolizer>
 <sld:Fill>
   <sld:GraphicFill>
     <sld:Graphic>
       <sld:Mark>
         ␣
↪ <sld:WellKnownName>shape://slash</sld:WellKnownName>
           <sld:Stroke>
             <sld:CssParameter␣
↪name="stroke">#0000ff</sld:CssParameter>
             <sld:CssParameter␣
↪name="stroke-linecap">round</sld:CssParameter>
             <sld:CssParameter␣
↪name="stroke-width">4</sld:CssParameter>
           </sld:Stroke>
         </sld:Mark>
         <sld:Size>8</sld:Size>
       </sld:Graphic>
   </sld:GraphicFill>
 </sld:Fill>
 <sld:VendorOption␣
↪name="random-seed">5</sld:VendorOption>
 <sld:VendorOption␣
↪name="random">grid</sld:VendorOption>
```

```
    <sld:VendorOption␣
↪name="random-tile-size">100</sld:VendorOption>
    <sld:VendorOption␣
↪name="random-rotation">free</sld:VendorOption>
    <sld:VendorOption␣
↪name="random-symbol-count">50</sld:VendorOption>
  </sld:PolygonSymbolizer>
```

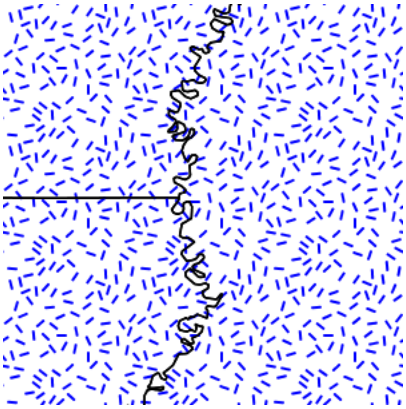


Fig. 6.85: *Random distribution of a diagonal line*

Randomized distributions can also be used for thematic mapping, for example, here is the SLD for a version of topp:states that displays the number of inhabitantìs varying the density of a random point distribution:

```
<?xml version="1.0" encoding="UTF-8"?>
<sld:UserStyle␣
↪xmlns="http://www.opengis.net/sld" xmlns:sld="http:/
↪/www.opengis.net/sld" xmlns:ogc="http://www.opengis.
↪net/ogc" xmlns:gml="http://www.opengis.net/gml">
  <sld:Name>Default Styler</sld:Name>
  <sld:FeatureTypeStyle>
    <sld:Name>name</sld:Name>
    <sld:Rule>
      <ogc:Filter>
        <ogc:And>
          <ogc:Not>
            <ogc:PropertyIsLessThan>
␣
↪            <ogc:PropertyName>PERSONS</ogc:PropertyName>
              <ogc:Literal>2000000</ogc:Literal>
            </ogc:PropertyIsLessThan>
          </ogc:Not>
          <ogc:Not>
            <ogc:PropertyIsGreaterThanOrEqualTo>
␣
↪            <ogc:PropertyName>PERSONS</ogc:PropertyName>
              <ogc:Literal>4000000</ogc:Literal>
            </ogc:PropertyIsGreaterThanOrEqualTo>
          </ogc:Not>
        </ogc:And>
      </ogc:Filter>
```

```
      <sld:PolygonSymbolizer>
        <sld:Fill>
          <sld:GraphicFill>
            <sld:Graphic>
              <sld:Mark>
↪               <sld:WellKnownName>circle</sld:WellKnownName>
                  <sld:Fill>
                    <sld:CssParameter␣
↪name="fill">#a9a9a9</sld:CssParameter>
                  </sld:Fill>
              </sld:Mark>
              <sld:Size>2</sld:Size>
            </sld:Graphic>
          </sld:GraphicFill>
        </sld:Fill>
        <sld:VendorOption␣
↪name="random">grid</sld:VendorOption>
        <sld:VendorOption␣
↪name="random-tile-size">100</sld:VendorOption>
        <sld:VendorOption␣
↪name="random-symbol-count">150</sld:VendorOption>
      </sld:PolygonSymbolizer>
      <sld:LineSymbolizer>
        <sld:Stroke/>
      </sld:LineSymbolizer>
    </sld:Rule>
    <sld:Rule>
      <ogc:Filter>
        <ogc:PropertyIsLessThan>
          <ogc:PropertyName>PERSONS</ogc:PropertyName>
          <ogc:Literal>2000000</ogc:Literal>
        </ogc:PropertyIsLessThan>
      </ogc:Filter>
      <sld:PolygonSymbolizer>
        <sld:Fill>
          <sld:GraphicFill>
            <sld:Graphic>
              <sld:Mark>
↪               <sld:WellKnownName>circle</sld:WellKnownName>
                  <sld:Fill>
                    <sld:CssParameter␣
↪name="fill">#a9a9a9</sld:CssParameter>
                  </sld:Fill>
              </sld:Mark>
              <sld:Size>2</sld:Size>
            </sld:Graphic>
          </sld:GraphicFill>
        </sld:Fill>
        <sld:VendorOption␣
↪name="random">grid</sld:VendorOption>
        <sld:VendorOption␣
↪name="random-tile-size">100</sld:VendorOption>
        <sld:VendorOption␣
↪name="random-symbol-count">50</sld:VendorOption>
      </sld:PolygonSymbolizer>
      <sld:LineSymbolizer>
```

```
            <sld:Stroke/>
          </sld:LineSymbolizer>
        </sld:Rule>
        <sld:Rule>
          <ogc:Filter>
            <ogc:PropertyIsGreaterThanOrEqualTo>
              <ogc:PropertyName>PERSONS</ogc:PropertyName>
              <ogc:Literal>4000000</ogc:Literal>
            </ogc:PropertyIsGreaterThanOrEqualTo>
          </ogc:Filter>
          <sld:PolygonSymbolizer>
            <sld:Fill>
              <sld:GraphicFill>
                <sld:Graphic>
                  <sld:Mark>
                    <sld:WellKnownName>circle</sld:WellKnownName>
                      <sld:Fill>
                        <sld:CssParameter name="fill">#a9a9a9</sld:CssParameter>
                      </sld:Fill>
                    </sld:Mark>
                    <sld:Size>2</sld:Size>
                  </sld:Graphic>
                </sld:GraphicFill>
              </sld:Fill>
              <sld:VendorOption name="random">grid</sld:VendorOption>
              <sld:VendorOption name="random-tile-size">100</sld:VendorOption>
              <sld:VendorOption name="random-symbol-count">500</sld:VendorOption>
            </sld:PolygonSymbolizer>
            <sld:LineSymbolizer>
              <sld:Stroke/>
            </sld:LineSymbolizer>
          </sld:Rule>
        </sld:FeatureTypeStyle>
      </sld:UserStyle>
```

Fig. 6.86: *Thematic map via point density approach*

Color compositing and color blending

It is possible to perform color blending and compositing, either between feature type styles or by associating blending operations with each symbolizer.

GeoServer implements most of the color compositing and blending modes suggested by the SVG compositing and blending level 1 specification. Either set of operations allows one to control how two overlapping layers/symbols are merged together to form a final map (as opposed to the normal behavior of just stacking images on top of each other).

This section will use the following definitions for the common terms "source" and "destination":

- **Source** : Image currently being painted *on top of* the map
- **Destination**: *Background* image that the source image is being drawn on

Specifying compositing and blending in SLD

Composites

Both compositing and blending can be specified in SLD by adding the following VendorOption to either the end of a Symbolizer or FeatureTypeStyle:

```
<VendorOption name="composite">multiply</VendorOption>
```

In case a custom opacity is desired, it can be added after the operation name separated with a comma:

```
<VendorOption␣
↪name="composite">multiply, 0.5</VendorOption>
```

Note: See the *full list of available modes*.

Warning: Blending against symbolizers causes exceptions inside the JDK when using OpenJDK. The issue is known and has been reportedly fixed, but only in OpenJDK 9.

One way to get around this issue with OpenJDK 7/8 is to install the Marlin renderer. This replaces the OpenJDK core renderer and does not suffer from the same issue.

Oracle JDK 7 or 8 does not show this issue.

Composite bases

For FeatureTypeStyles an additional vendor option can be added to control compositing groups:

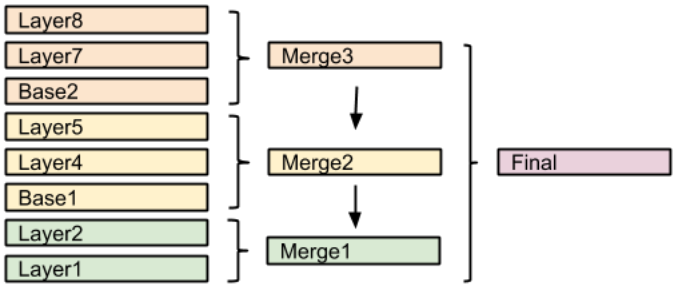
```
<VendorOption name="composite-base">true</VendorOption>
```

This will tell the rendering engine to use that FeatureTypeStyle as the destination, and will compose all subsequent FeatureTypeStyle/Layers on top of it, until another base is found. Once the full set of layers against a base is composed, then the base itself will be composed against the next set of composed layers, using its own compositing operator, if present.

Without this setting, the destination will be the full stack of all previous FeatureTypeStyles and layers drawn before the current one. This can be limiting for two reasons:

- It limits the usefulness of alpha-composite masking operations
- It breaks the WMS model, where the client can decide freely how to stack layers (the desired compositing effects will be achieved only when a certain stack of layers is used)

Consider the following example:



In this example, the first two layers are drawn on top of each other, forming "Merge1".

The third layer is a composite base, as such it won't be merged on top of the already drawn map immediately, but it will be drawn to an off-screen buffer, and layer 4 and 5 will be drawn/composited on top of it. Once that happens, "Merge2" is ready, and gets drawn on top of "Merge1",

The next layer is another base, so "Base2" will be again drawn to an off-screen buffer, and layer 7 and 8 will be drawn/composited on top of it, forming Merge3. Once Merge3 is ready, it will be drawn/composited on top of the already fused Merge1/Merge2, generating the final map.

Composite and blending modes

There are two types of modes: alpha composite and color blending.

Alpha compositing controls how two images are merged together by using the alpha levels of the two. No color mixing is being performed, only pure binary selection (either one or the other).

Color blending modes mix the colors of source and destination in various ways. Each pixel in the result will be some sort of combination between the source and destination pixels.

The following page shows the full list of available modes. (See the *syntax* page for more details.) To aid in comprehension, two source and two destination images will be used to show visually how each mode works:

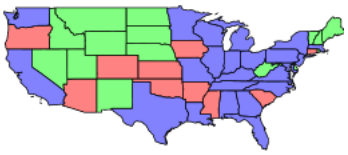
Source 1	Source 2
	

Destination 1	Destination 2
	

Alpha compositing modes

copy

Only the source will be present in the output.

Example 1	Example 2
	

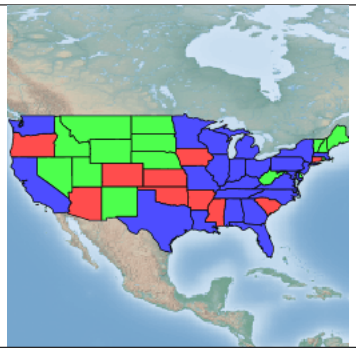
destination

Only the destination will be present in the output

Example 1	Example 2
	

source-over

The source is drawn over the destination, and the destination is visible where the source is transparent. Opposite of *destination-over*.

Example 1	Example 2
	

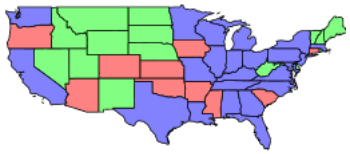
destination-over

The source is drawn below the destination, and is visible only when the destination is transparent. Opposite of *source-over*.

Example 1	Example 2
	

source-in

The source is visible only when overlapping some non-transparent pixel of the destination. This allows the background map to act as a mask for the layer/feature being drawn. Opposite of *destination-in*.

Example 1	Example 2
	

destination-in

The destination is retained only when overlapping some non transparent pixel in the source. This allows the layer/feature to be drawn to act as a mask for the background map. Opposite of *source-in*.

Example 1	Example 2
	

source-out

The source is retained only in areas where the destination is transparent. This acts as a reverse mask when compared to *source-in*.

Example 1	Example 2
	

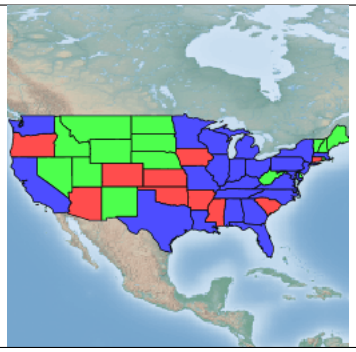
destination-out

The destination is retained only in areas where the source is transparent. This acts as a reverse mask when compared to *destination-in*.

Example 1	Example 2
	

source-atop

The destination is drawn fully, while the source is drawn only where it intersects the destination.

Example 1	Example 2
	

destination-atop

The source is drawn fully, and the destination is drawn over the source and only where it intersects it.

Example 1	Example 2
	

xor

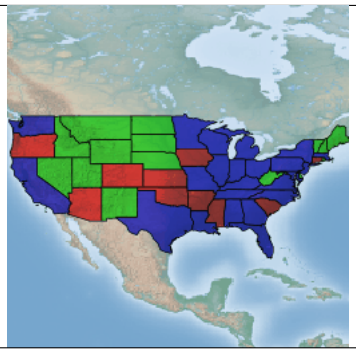
"Exclusive Or" mode. Each pixel is rendered only if either the source or the destination is not blank, but not both.

Example 1	Example 2
	

Color blending modes

multiply

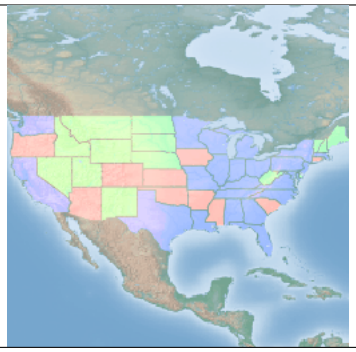
The source color is multiplied by the destination color and replaces the destination. The resulting color is always at least as dark as either the source or destination color. Multiplying any color with black results in black. Multiplying any color with white preserves the original color.

Example 1	Example 2
	

screen

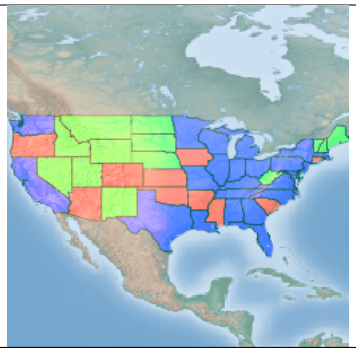
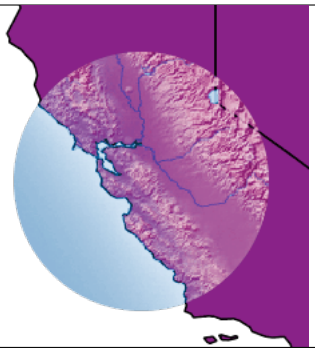
Multiplies the complements of the source and destination color values, then complements the result. The end result color is always at least as light as either of the two constituent colors. Screening any color with white produces white; screening with black leaves the original color unchanged.

The effect is similar to projecting multiple photographic slides simultaneously onto a single screen.

Example 1	Example 2
	

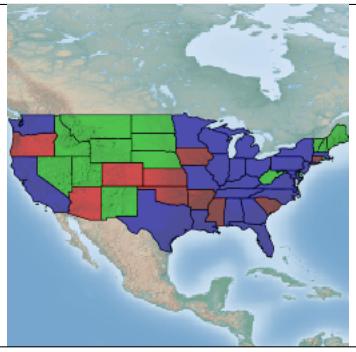
overlay

Multiplies (screens) the colors depending on the destination color value. Source colors overlay the destination while preserving its highlights and shadows. The backdrop color is not replaced but is mixed with the source color to reflect the lightness or darkness of the backdrop.

Example 1	Example 2
	

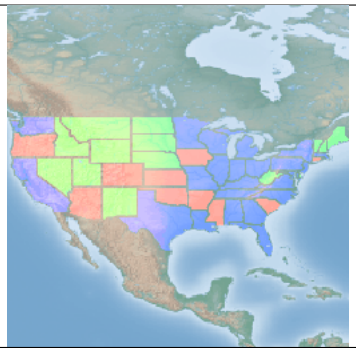
darken

Selects the darker of the destination and source colors. The destination is replaced with the source only where the source is darker.

Example 1	Example 2
	

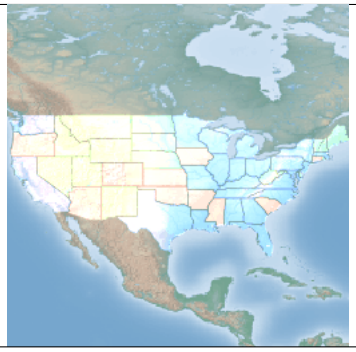
lighten

Selects the lighter of the destination and source colors. The destination is replaced with the source only where the source is lighter.

Example 1	Example 2
	

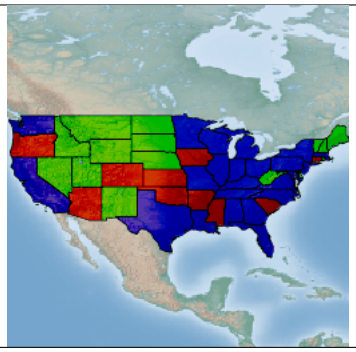
color-dodge

Brightens the destination color to reflect the source color. Drawing with black produces no changes.

Example 1	Example 2
	

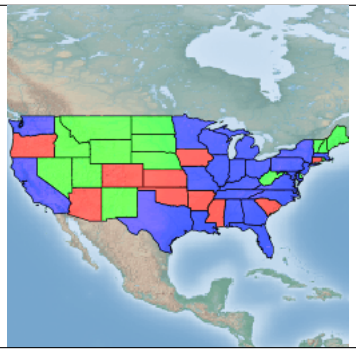
color-burn

Darkens the destination color to reflect the source color. Drawing with white produces no change.

Example 1	Example 2
	

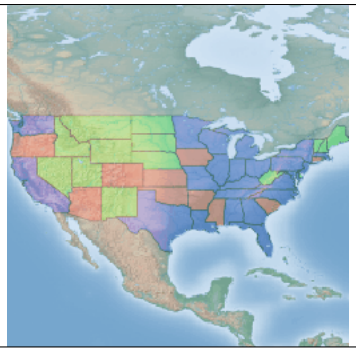
hard-light

Multiplies or screens the colors, depending on the source color value. The effect is similar to shining a harsh spotlight on the destination.

Example 1	Example 2
	

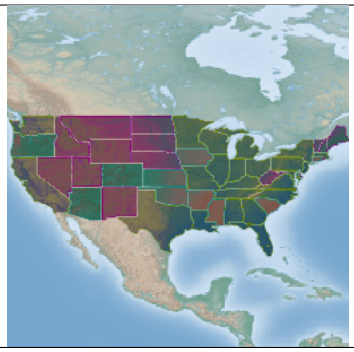
soft-light

Darkens or lightens the colors, depending on the source color value. The effect is similar to shining a diffused spotlight on the destination.

Example 1	Example 2
	

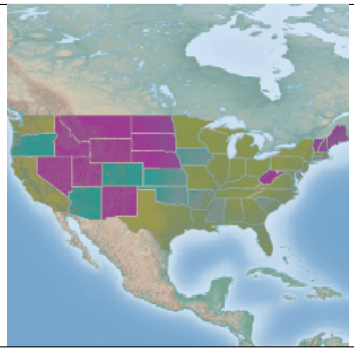
difference

Subtracts the darker of the two constituent colors from the lighter color. White inverts the destination color; black produces no change.

Example 1	Example 2
	

exclusion

Produces an effect similar to that of *difference* but lower in contrast. White inverts the destination color; black produces no change.

Example 1	Example 2
	

Compositing and blending example

Let's say we want to draw the `topp:states` layer so that the polygons are not filled with the population keyed colors, but only an inner border inside the polygon should appear, leaving the internal fully transparent.

This is the destination:

Using alpha blending, this can be achived by creating a mask around the state borders with a thick stroke, and then using a "destination-in" alpha compositing.

This is the source (mask):

The SLD will contain three FeatureTypeStyles. The first one would be the standard rules (states colored by population) and the last one

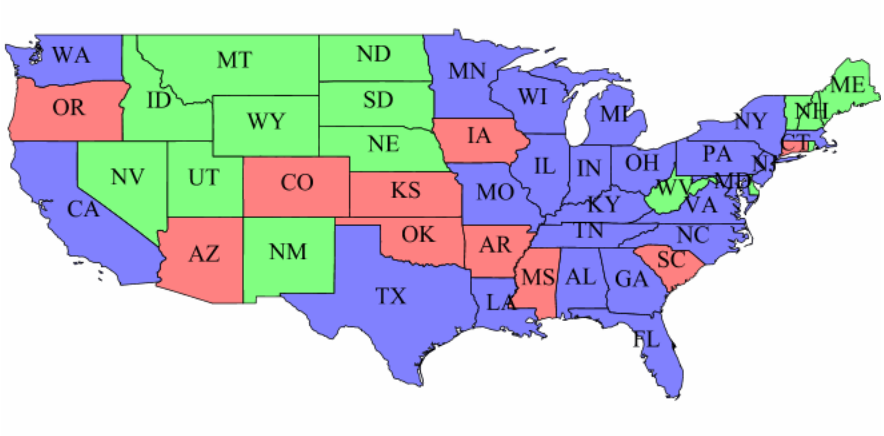


Fig. 6.87: topp:states layer

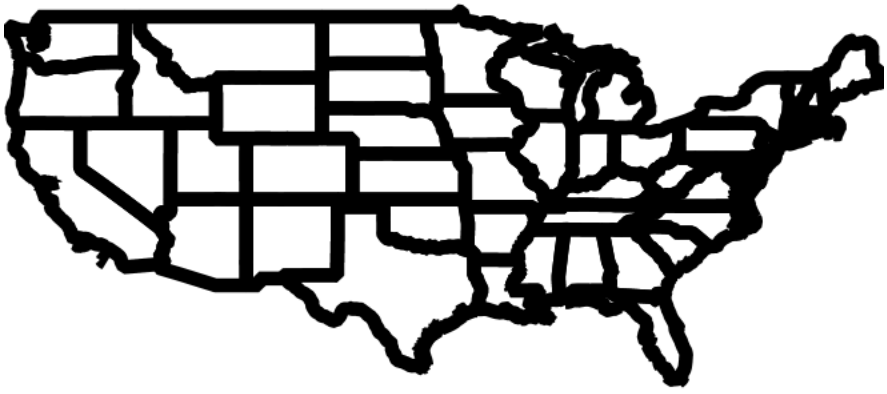
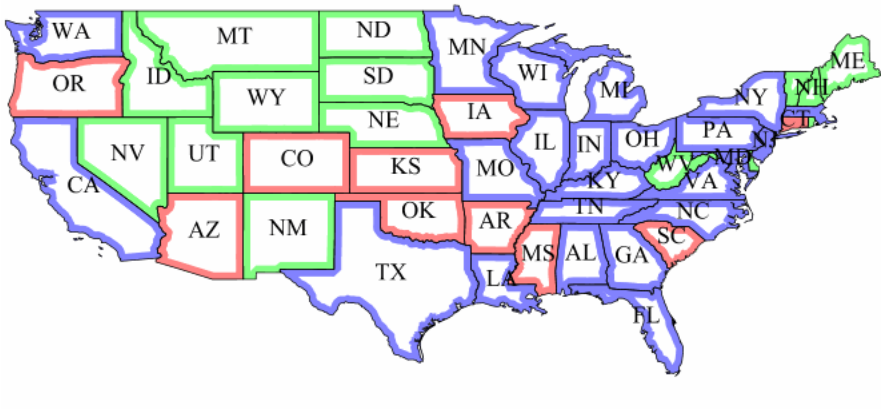


Fig. 6.88: Layer mask

will contain the label rules. The second (middle) one is where the blending will occur:

```
...
<FeatureTypeStyle>
  <!-- Usual states rules, skipped for brevity -->
</FeatureTypeStyle>
<FeatureTypeStyle>
  <Rule>
    <LineSymbolizer>
      <Stroke>
↪   <CssParameter name="stroke-width">10</CssParameter>
↪    <CssParameter name="stroke">#000000</CssParameter>
      </Stroke>
    </LineSymbolizer>
  </Rule>
  <VendorOption
↪name="composite">destination-in</VendorOption>
</FeatureTypeStyle>
<FeatureTypeStyle>
  <!-- The label rules, skipped for brevity -->
</FeatureTypeStyle>
...
```

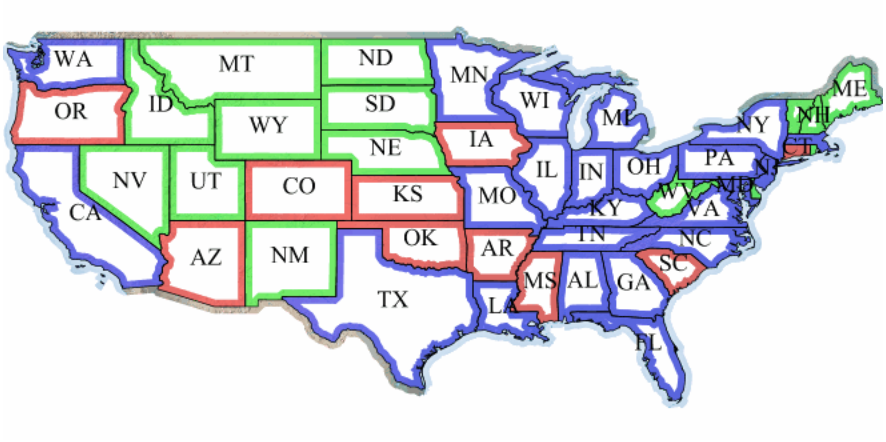
This is the result of the composition:



Now, if for example someone makes a WMS call in which the another layer is drawn below this one, the result will look like this:

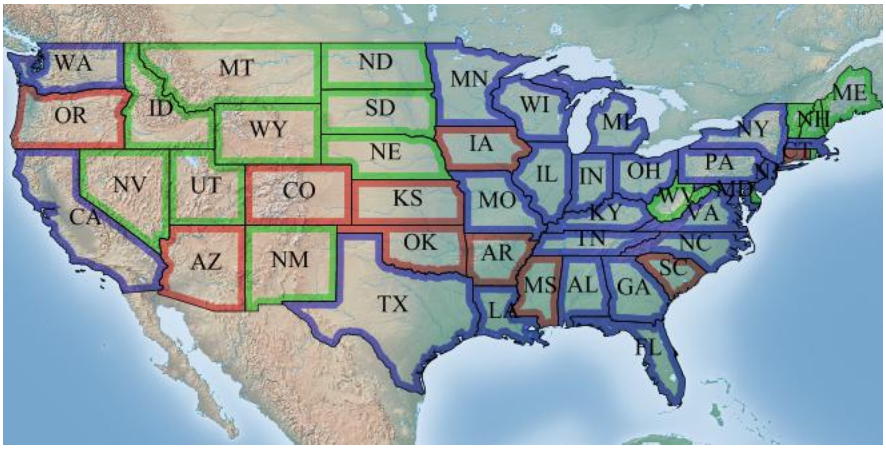
This other background layer is hardly visible, because it has been cut by the mask. This shows the risks of using alpha compositing without care in a WMS setting.

In order to achieve the desired result no matter how the client composes the request, the first FeatureTypeStyle that draws the polygons (the states themselves) needs to be set as a *compositing base*, ensuring the mask will only be applied to it.




```
<VendorOption name="composite-base">true</VendorOption>
```

The result will look like the following (though a multiply blend was added to the base to ensure a nice visual transparency effect on the border lines):



`Download the final style`

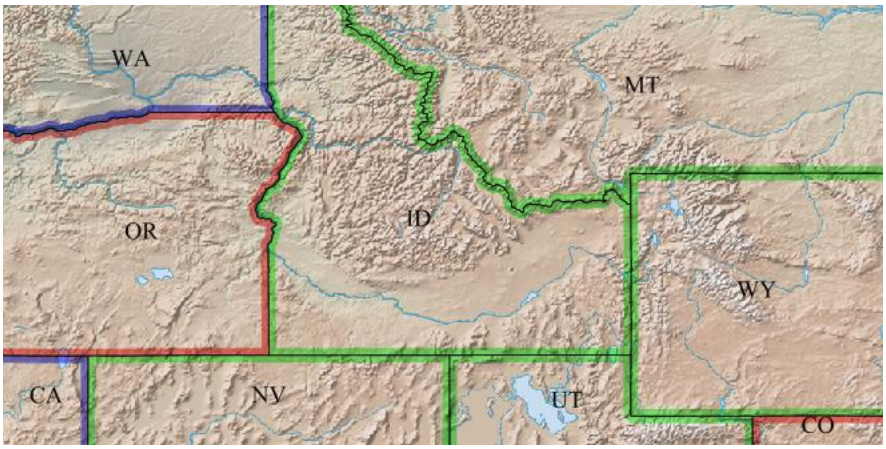
Note: See the *full list of available modes*.

Z ordering features within and across feature types and layers

Starting with GeoServer 2.8.0 it is possible to control the order in which the features are being loaded and painted on the map, thus replicating the same above/below relationships found in the reality.

Enabling z-ordering in a single FeatureTypeStyle

The z-ordering is implemented as a new FeatureTypeStyle vendor option, `sortBy`, which controls in which order the features are ex-



tracted from the data source, and thus painted. The `sortBy` syntax is the same as the WFS one, that is, a list of comma separated field names, with an optional direction modifier (ascending being the default):

```
field1 [A|D], field2 [A|D], ... , fieldN [A|D]
```

Some examples:

- "z": sorts the features based on the `z` field, ascending (lower z values are painted first, higher later)
- "cat,z D": sorts the features on the `cat` attribute, with ascending order, and for those that have the same `cat` value, the sorting is on descending `z`
- "cat D,z D": sorts the features on the `cat` attribute, with descending order, and for those that have the same `cat` value, the sorting is on descending `z`

So, if we wanted to order features based on a single "elevation" attribute we'd be using the following SLD snippet:

```
...
<sld:FeatureTypeStyle>
  <sld:Rule>
    ...
    <!-- filters and symbolizers here -->
    ...
  </sld:Rule>
  <sld:VendorOption␣
↪name="sortBy">elevation</sld:VendorOption>
</sld:FeatureTypeStyle>
...
```

z-ordering across FeatureTypeStyle

It is a common need to perform road casing against a complex road network, which can have its own z-ordering needs (e.g., over and under passes). Casing is normally achieved by using two separate

two `FeatureTypeStyle`, one drawing a thick line, one drawing a thin one.

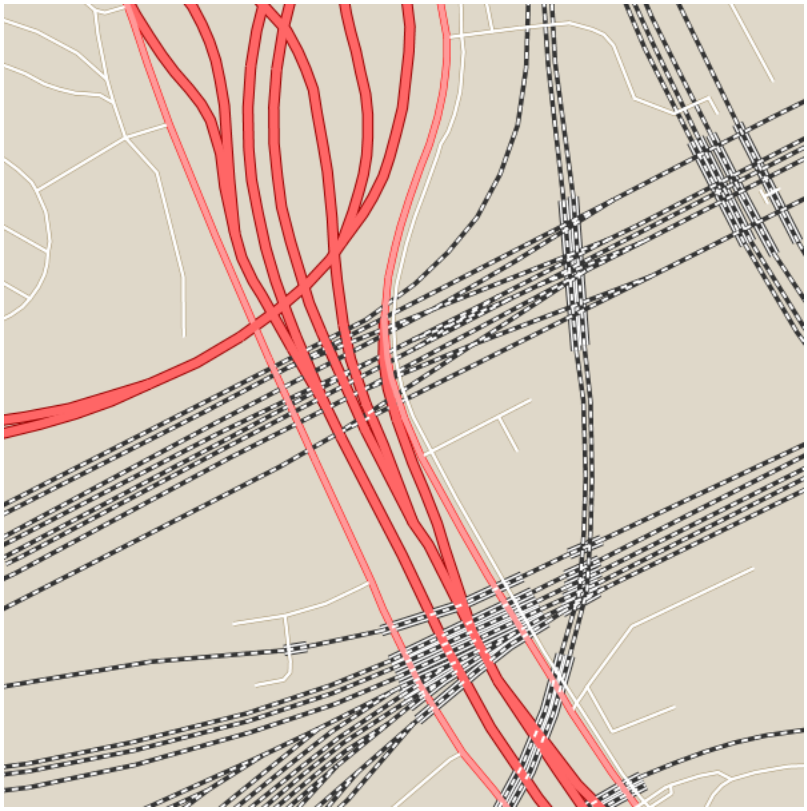
Let's consider a simple data set, made of just three roads:

```
_=geom:LineString:404000,z:int
Line.1=LINESTRING(0 4, 10 4)|1
Line.2=LINESTRING(0 6, 10 6)|3
Line.3=LINESTRING(7 0, 7 10)|1
```

Adding a "sortBy" rule to both `FeatureTypeStyle` objects will achieve no visible result:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<StyledLayerDescriptor version="1.0.0"
  xsi:schemaLocation="http:/
↪/www.opengis.net/sld StyledLayerDescriptor.xsd"
  xmlns="http://www.opengis.
↪net/sld" xmlns:ogc="http://www.opengis.net/ogc"
  xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xsi=
↪"http://www.w3.org/2001/XMLSchema-instance">
  <!-- a named layer␣
↪is the basic building block of an sld document -->

  <NamedLayer>
    <UserStyle>
      <FeatureTypeStyle>
        <Rule>
          <LineSymbolizer>
            <Stroke>

↪    <CssParameter name="stroke">#FF0000</CssParameter>

↪    <CssParameter name="stroke-width">8</CssParameter>
            </Stroke>
          </LineSymbolizer>
        </Rule>

↪ <sld:VendorOption name="sortBy">z</sld:VendorOption>
      </FeatureTypeStyle>
      <FeatureTypeStyle>
        <Rule>
          <LineSymbolizer>
            <Stroke>

↪    <CssParameter name="stroke">#FFFFFF</CssParameter>

↪    <CssParameter name="stroke-width">6</CssParameter>
            </Stroke>
          </LineSymbolizer>
        </Rule>

↪ <sld:VendorOption name="sortBy">z</sld:VendorOption>
      </FeatureTypeStyle>
    </UserStyle>
  </NamedLayer>
</StyledLayerDescriptor>
```

The result will be the following:



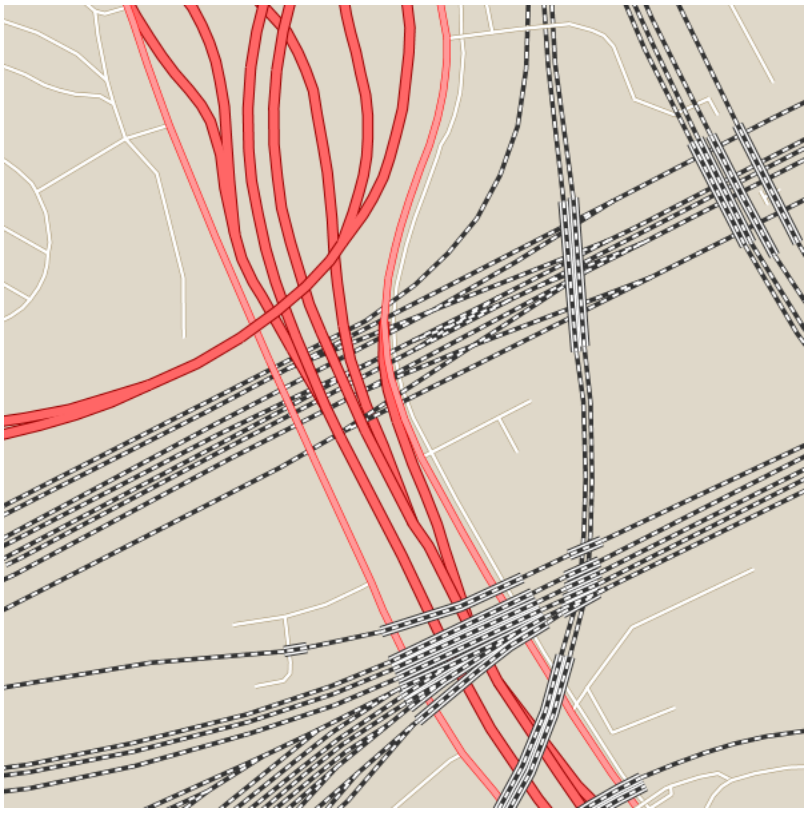
This is happening because while the roads are loaded in the right order, `Line.1`, `Line.3`, `Line.2`, they are all painted with the tick link first, and then the code will start over, and paint them all with the thin line.

In order to get both casing and z-ordering to work a new vendor option, `sortByGroup`, needs to be added to both `FeatureTypeStyle`, grouping them in a single z-ordering paint.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<StyledLayerDescriptor version="1.0.0"
  xsi:schemaLocation="http:/
↪/www.opengis.net/sld StyledLayerDescriptor.xsd"
  xmlns="http://www.opengis.
↪net/sld" xmlns:ogc="http://www.opengis.net/ogc"
  xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xsi=
↪"http://www.w3.org/2001/XMLSchema-instance">
  <!-- a named layer␣
↪is the basic building block of an sld document -->

  <NamedLayer>
    <UserStyle>
      <FeatureTypeStyle>
        <Rule>
          <LineSymbolizer>
            <Stroke>

↪    <CssParameter name="stroke">#FF0000</CssParameter>

↪    <CssParameter name="stroke-width">8</CssParameter>
```

```
                    </Stroke>
                  </LineSymbolizer>
                </Rule>

↪ <sld:VendorOption name="sortBy">z</sld:VendorOption>
                <sld:VendorOption␣
↪name="sortByGroup">roads</sld:VendorOption>
              </FeatureTypeStyle>
              <FeatureTypeStyle>
                <Rule>
                  <LineSymbolizer>
                    <Stroke>

↪   <CssParameter name="stroke">#FFFFFF</CssParameter>

↪   <CssParameter name="stroke-width">6</CssParameter>
                    </Stroke>
                  </LineSymbolizer>
                </Rule>

↪ <sld:VendorOption name="sortBy">z</sld:VendorOption>
                <sld:VendorOption␣
↪name="sortByGroup">roads</sld:VendorOption>
              </FeatureTypeStyle>
            </UserStyle>
          </NamedLayer>
        </StyledLayerDescriptor>
```

The result will be the following:



When grouping is used, the code will first paint `Line.1,Line3` with the thick line, then track back and paint them with the thin line, then move to paint `Line.2` with the thick line, and finally `Line.2` with the thin line, achieving the desired result.

z-ordering across layers

Different layers, such for example roads and rails, can have their features z-ordered together by putting all the `FeatureTypeStyle` in their styles in the same `sortByGroup`, provided the following conditions are met:

- The layers are side by side in the WMS request/layer group. In other words, the z-ordering allows to break the WMS specified order only if the layers are directly subsequent in the request. This can be extended to any number of layers, provided the progression of `FeatureTypeStyle` in the same group is not broken
- There is no FeatureTypeStyle in the layer style that's breaking the sequence

  Let's consider an example, with a rails layer having two `FeatureTypeStyle`, one with a group, the other not:

FeatureTypeStyle id	SortByGroup id
rails1	linework
rails2	`none`

We then have a roads layer with two `FeatureTypeStyle`, both in the same group:

FeatureTypeStyle id	SortByGroup id
road1	linework
road2	linework

If the WMS request asks for `&layers=roads,rails`, then the expanded `FeatureTypeStyle` list will be:

FeatureTypeStyle id	SortByGroup id
road1	linework
road2	linework
rails1	linework
rails2	`none`

As a result, the `road1,road2,rails1` will form a single group, and this will result in the rails be merged with the roads when z-ordering.

If instead the WMS request asks for *&layers=rails,roads'*, then the expanded `FeatureTypeStyle` list will be:

FeatureTypeStyle id	SortByGroup id
rails1	linework
rails2	none
road1	linework
road2	linework

The `rails2` feature type style breaks the sequence, as a result, the rails will not be z-ordered in the same group as the roads.

Z ordering single layer example

The OpenStreetMap dataset uses extensively a `z_order` attribute to model the above/below relationships between elements in the real world.

A `small downloadable shapefile` is provided that shows a small area with a rich set of different z-orders, where roads and rails go above and below each other. For reference, this is the dataset schema:

Name	Type	Notes
osm_id	numeric	
type	string	The type of the segment, can be "mainroads", "minorroads", "railways", ...
bridge	numeric	0 or 1
ref	numeric	0 or 1
tunnel	numeric	
oneway	numeric	0 or 1
z_order	numeric	
class	string	

The dataset contains several different values for z_order, in particular: -10, -7, -5, -3, -1, 0, 3, 4, 5, 7, 9, 10, 13, 14, 15, 17, 19.

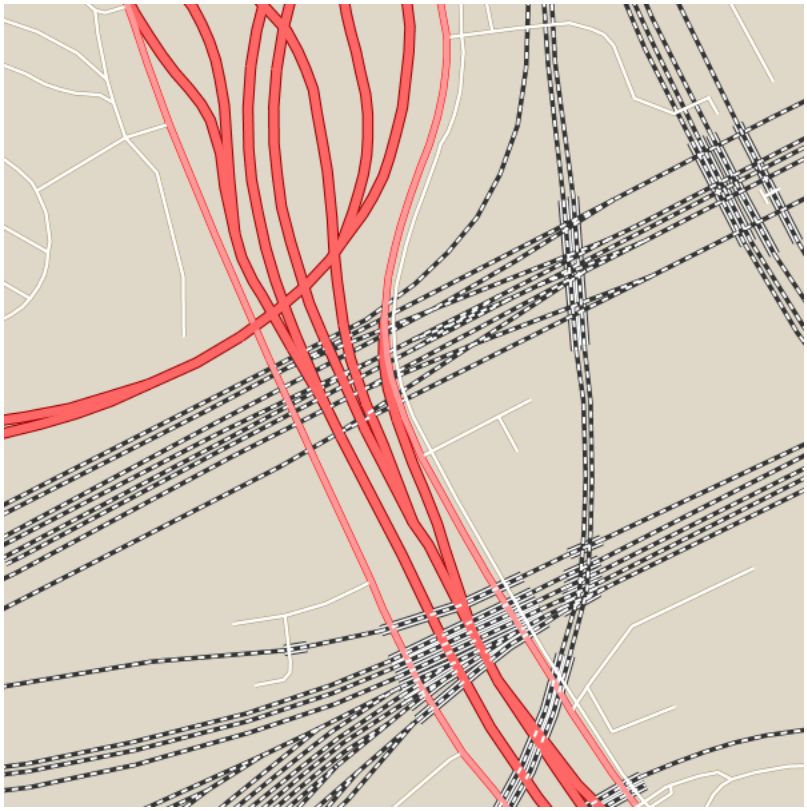
Here is a sample CSS style using z-ordering, but not groups, to perform the display. Road casing is achieved by multiple FeatureTypeStyle, or `z-index` values in CSS:

```
[class = 'railways' and bridge = 1] {
  stroke: #333333;
  stroke-width: 8;
  z-index: 0;
}

[class = 'minorroads'] {
  stroke: #a69269;
  stroke-width: 3;
  z-index: 0;
}

[class = 'mainroads'] {
  stroke: #ff0000;
  stroke-width: 5;
```

```
  z-index: 0;
}

[class = 'motorways'] {
  stroke: #990000;
  stroke-width: 8;
  z-index: 0;
}

[class = 'railways' and bridge = 1] {
  stroke: #ffffff;
  stroke-width: 6;
  z-index: 1;
}

[class = 'railways'] {
  stroke: #333333;
  stroke-width: 3;
  z-index: 2;
}

[class = 'railways'] {
  stroke: #ffffff;
  stroke-width: 1.5;
  stroke-dasharray: 5, 5;
  z-index: 3;
}

[class = 'motorways'] {
  stroke: #ff6666;
  stroke-width: 6;
  stroke-linecap: round;
  z-index: 3;
}

[class = 'minorroads'] {
  stroke: #ffffff;
  stroke-width: 2,5;
  stroke-linecap: round;
  z-index: 3;
}

[class = 'mainroads'] {
  stroke: #ff9999;
  stroke-width: 4;
  stroke-linecap: round;
  z-index: 3;
}

* {
 sort-by: "z_order";
}
```

The sorting is achieved by using the `sort-by` property, which translates into a `sortBy` VendorOption in SLD. A full equivalent SLD is `available for download`.

This is the resulting map:



As one can see, there are evident issues:

- Roads and rails are not showing any evident z-ordering, in fact, all rails are below roads, but their dashed white center shows a mix of below and above roads
- The rails bridges (depicted with a third thicker line around the rail symbol) are consistently below some other road or rail, while they should be above.

This is mostly happening because the various FeatureTypeStyle elements are not put doctor in a single group.

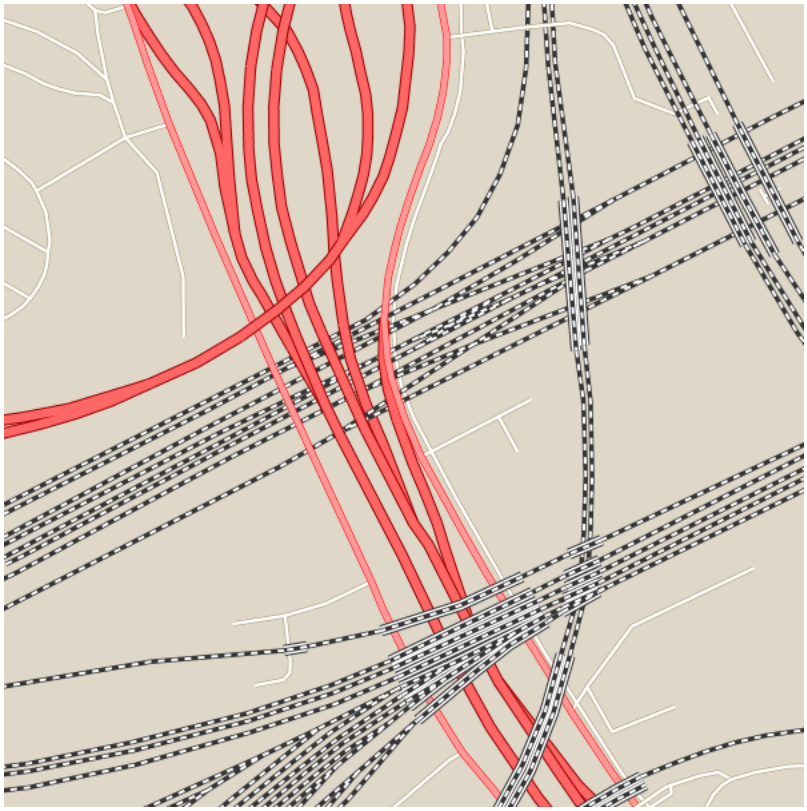
A slight change in the CSS, grouping all levels in the same sortByGroup, solves the issues above:

```
[class = 'railways' and bridge = 1] {
  stroke: #333333;
  stroke-width: 8;
  z-index: 0;
}

[class = 'minorroads'] {
  stroke: #a69269;
  stroke-width: 3;
  z-index: 0;
}

[class = 'mainroads'] {
  stroke: #ff0000;
```

```
  stroke-width: 5;
  z-index: 0;
}

[class = 'motorways'] {
  stroke: #990000;
  stroke-width: 8;
  z-index: 0;
}

[class = 'railways' and bridge = 1] {
  stroke: #ffffff;
  stroke-width: 6;
  z-index: 1;
}

[class = 'railways'] {
  stroke: #333333;
  stroke-width: 3;
  z-index: 2;
}

[class = 'railways'] {
  stroke: #ffffff;
  stroke-width: 1.5;
  stroke-dasharray: 5, 5;
  z-index: 3;
}

[class = 'motorways'] {
  stroke: #ff6666;
  stroke-width: 6;
  stroke-linecap: round;
  z-index: 3;
}

[class = 'minorroads'] {
  stroke: #ffffff;
  stroke-width: 2,5;
  stroke-linecap: round;
  z-index: 3;
}

[class = 'mainroads'] {
  stroke: #ff9999;
  stroke-width: 4;
  stroke-linecap: round;
  z-index: 3;
}

* {
 sort-by: "z_order";
 sort-by-group: "roadsGroup";
}
```

A full equivalent SLD is also `available for download`.

The result now shows proper z-ordering:



6.2.6 SLD Tips and Tricks

This section details various advanced strategies for working with SLD.

Styling mixed geometry types

On occasion one might need to style a geometry column whose geometry type can be different for each feature (some are polygons, some are points, etc), and use different styling for different geometry types.

SLD 1.0 does not provide a clean solution for dealing with this situation. Point, Line, and Polygon symbolizers do not select geometry by type, since each can apply to all geometry types:

- Point symbolizers apply to any kind of geometry. If the geometry is not a point, the centroid of the geometry is used.
- Line symbolizers apply to both lines and polygons. For polygons the boundary is styled.
- Polygon symbolizers apply to lines, by adding a closing segment connecting the first and last points of the line.

  There is also no standard filter predicate to identify geometry type which could be used in rules.

  This section suggests a number of ways to accomplish styling by

geometry type. They require either data restructuring or the use of non-standard filter functions.

Restructuring the data

There are a few ways to restructure the data so that it can be styled by geometry type using only standard SLD constructs.

Split the table

The first and obvious one is to split the original table into a set of separate tables, each one containing a single geometry type. For example, if table `findings` has a geometry column that can contain point, lines, and polygons, three tables can be created, each one containing a single geometry type.

Separate geometry columns

A second way is to use one table and separate geometry columns. So, if the table `findings` has a `geom` column, the restructured table will have `point`, `line` and `polygon` columns, each of them containing just one geometry type. After the restructuring, the symbolizers will refer to a specific geometry, for example:

```
<PolygonSymbolizer>
    <Geometry><ogc:PropertyName>
↪polygon</ogc:PropertyName></Geometry>
</PolygonSymbolizer>
```

This way each symbolizer will match only the geometry types it is supposed to render, and skip over the rows that contain a null value.

Add a geometry type column

A third way is to add a geometry type column allowing standard filtering constructs to be used, and then build a separate rule per geometry type. In the example above a new attribute, `gtype` will be added containing the values `Point`, `Line` and `Polygon`. The following SLD template can be used after the change:

```
<Rule>
   <ogc:Filter>
      <ogc:PropertyIsEqualTo>
         <ogc:PropertyName>gtype</ogc:PropertyName>
         <ogc:Literal>Point</ogc:Literal>
      </ogc:PropertyIsEqualTo>
   </ogc:Filter>
   <PointSymbolizer>
      ...
   </PointSymbolizer>
</Rule>
```

```
<Rule>
   <ogc:Filter>
      <ogc:PropertyIsEqualTo>
         <ogc:PropertyName>gtype</ogc:PropertyName>
         <ogc:Literal>Line</ogc:Literal>
      </ogc:PropertyIsEqualTo>
   </ogc:Filter>
   <LineSymbolizer>
      ...
   </LineSymbolizer>
</Rule>
<Rule>
   <ogc:Filter>
      <ogc:PropertyIsEqualTo>
         <ogc:PropertyName>gtype</ogc:PropertyName>
         <ogc:Literal>Polygon</ogc:Literal>
      </ogc:PropertyIsEqualTo>
   </ogc:Filter>
   <PolygonSymbolizer>
      ...
   </PolygonSymbolizer>
</Rule>
```

The above suggestions assume that restructuring the data is technically possible. This is usually true in spatial databases that provide functions that allow determining the geometry type.

Create views

A less invasive way to get the same results without changing the structure of the table is to create views that have the required structure. This allows the original data to be kept intact, and the views may be used for rendering.

Using SLD rules and filter functions

SLD 1.0 uses the OGC Filter 1.0 specification for filtering out the data to be styled by each rule. Filters can contain *Filter functions* to compute properties of geometric values. In GeoServer, filtering by geometry type can be done using the `geometryType` or `dimension` filter functions.

Note: The Filter Encoding specification provides a standard syntax for filter functions, but does not mandate a specific set of functions. SLDs using these functions may not be portable to other styling software.

geometryType function

The `geometryType` function takes a geometry property and returns a string, which (currently) is one of the values `Point`, `LineString`, `LinearRing`, `Polygon`, `MultiPoint`, `MultiLineString`, `MultiPolygon` and `GeometryCollection`.

Using this function, a `Rule` matching only single points can be written as:

```
<Rule>
   <ogc:PropertyIsEqualTo>
      <ogc:Function name="geometryType">
         <ogc:PropertyName>geom</ogc:PropertyName>
      </ogc:Function>
      <ogc:Literal>Point</ogc:Literal>
   </ogc:PropertyIsEqualTo>
   <PointSymbolizer>
     ...
   </PointSymbolizer>
</Rule>
```

The filter is more complex if it has to match all linear geometry types. In this case, it looks like:

```
<Rule>
   <ogc:Filter>
     <ogc:PropertyIsEqualTo>
       <ogc:Function name="in3">
          <ogc:Function name="geometryType">
              <ogc:PropertyName>geom</ogc:PropertyName>
          </ogc:Function>
          <ogc:Literal>LineString</ogc:Literal>
          <ogc:Literal>LinearRing</ogc:Literal>
          <ogc:Literal>MultiLineString</ogc:Literal>
       </ogc:Function>
       <ogc:Literal>true</ogc:Literal>
     </ogc:PropertyIsEqualTo>
   </ogc:Filter>
   <LineSymbolizer>
     ...
   </LineSymbolizer>
</Rule>
```

This filter is read as `geometryType(geom) in ("LineString", "LinearRing", "MultiLineString")`. Filter functions in Filter 1.0 have a fixed number of arguments, so there is a series of `in` functions whose names correspond to the number of arguments they accept: `in2`, `in3`, ..., `in10`.

dimension function

A slightly simpler alternative is to use the geometry `dimension` function to select geometries of a desired dimension. Dimension 0 selects Points and MultiPoints, dimension 1 selects LineStrings, LinearRings and MultiLineStrings, and dimension 2 selects Polygons and MultiPolygons. The following example shows how to select linear geometries:

```
<Rule>
   <ogc:PropertyIsEqualTo>
      <ogc:Function name="dimension">
         <ogc:PropertyName>geom</ogc:PropertyName>
```

```
            </ogc:Function>
            <ogc:Literal>1</ogc:Literal>
         </ogc:PropertyIsEqualTo>
         <LineSymbolizer>
            ...
         </LineSymbolizer>
      </Rule>
```

Styling using Transformation Functions

The Symbology Encoding 1.1 specification defines the following **transformation functions**:

- `Recode` transforms a set of discrete attribute values into another set of values
- `Categorize` transforms a continuous-valued attribute into a set of discrete values
- `Interpolate` transforms a continuous-valued attribute into another continuous range of values

These functions provide a concise way to compute styling parameters from feature attribute values. GeoServer implements them as *Filter functions* with the same names.

Note: The GeoServer function syntax is slightly different to the SE 1.1 definition, since the specification defines extra syntax elements which are not available in GeoServer functions.

These functions can make style documents more concise, since they express logic which would otherwise require many separate rules or complex Filter expressions, They even allow logic which is impossible to express any other way. A further advantage is that they often provide superior performance to explicit rules.

One disadvantage of using these functions for styling is that they are not displayed in WMS legend graphics.

Recode

The `Recode` filter function transforms a set of discrete values for an attribute into another set of values. The function can be used within SLD styling parameters to convert the value of a feature attribute into specific values for a parameter such as color, size, width, opacity, etc.

The recoding is defined by a set of *(input, output)* value pairs.

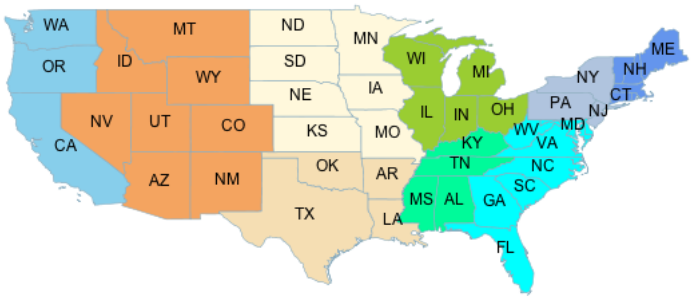
Example

Consider a chloropleth map of the US states dataset using the fill color to indicate the topographic regions for the states. The dataset has an attribute `SUB_REGION` containing the region code for each state. The `Recode` function is used to map each region code into a different color.

The symbolizer for this style is:

```
<PolygonSymbolizer>
   <Fill>
     <CssParameter name="fill">
       <ogc:Function name="Recode">
         <!-- Value to transform -->
         <ogc:Function name="strTrim">
 ↪
          <ogc:PropertyName>SUB_REGION</ogc:PropertyName>
         </ogc:Function>

         <!-- Map of input to output values -->
         <ogc:Literal>N Eng</ogc:Literal>
         <ogc:Literal>#6495ED</ogc:Literal>

         <ogc:Literal>Mid Atl</ogc:Literal>
         <ogc:Literal>#B0C4DE</ogc:Literal>

         <ogc:Literal>S Atl</ogc:Literal>
         <ogc:Literal>#00FFFF</ogc:Literal>

         <ogc:Literal>E N Cen</ogc:Literal>
         <ogc:Literal>#9ACD32</ogc:Literal>

         <ogc:Literal>E S Cen</ogc:Literal>
         <ogc:Literal>#00FA9A</ogc:Literal>

         <ogc:Literal>W N Cen</ogc:Literal>
         <ogc:Literal>#FFF8DC</ogc:Literal>

         <ogc:Literal>W S Cen</ogc:Literal>
         <ogc:Literal>#F5DEB3</ogc:Literal>

         <ogc:Literal>Mtn</ogc:Literal>
         <ogc:Literal>#F4A460</ogc:Literal>

         <ogc:Literal>Pacific</ogc:Literal>
         <ogc:Literal>#87CEEB</ogc:Literal>
       </ogc:Function>
     </CssParameter>
   </Fill>
</PolygonSymbolizer>
```

This style produces the following output:



Categorize

The `Categorize` filter function transforms a continuous-valued attribute into a set of discrete values. The function can be used within SLD styling parameters to convert the value of a feature attribute into specific values for a parameter such as color, size, width, opacity, etc.

The categorization is defined by a list of alternating output values and data thresholds. The threshold values define the breaks between the input ranges. Inputs are converted into output values depending on which range they fall in.

Example

Consider a chloropleth map of the US states dataset using the fill color to indicate a categorization of the states by population. The dataset has attributes `PERSONS` and `LAND_KM` from which the population density is computed using the `Div` operator. This value is input to the `Categorize` function, which is used to assign different colors to the density ranges [<= 20], [20 - 100], and [> 100].

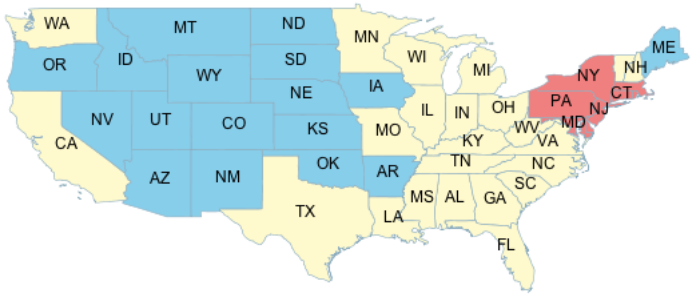
The symbolizer for this style is:

```
<PolygonSymbolizer>
   <Fill>
     <CssParameter name="fill">
       <ogc:Function name="Categorize">
         <!-- Value to transform -->
         <ogc:Div>
           <ogc:PropertyName>PERSONS</ogc:PropertyName>
           <ogc:PropertyName>LAND_KM</ogc:PropertyName>
         </ogc:Div>

         <!-- Output values and thresholds -->
         <ogc:Literal>#87CEEB</ogc:Literal>
         <ogc:Literal>20</ogc:Literal>
         <ogc:Literal>#FFFACD</ogc:Literal>
         <ogc:Literal>100</ogc:Literal>
         <ogc:Literal>#F08080</ogc:Literal>

       </ogc:Function>
     </CssParameter>
   </Fill>
</PolygonSymbolizer>
```

This style produces the following output:

Interpolate

The `Interpolate` filter function transforms a continuous-valued attribute into another continuous range of values. The function can be used within SLD styling parameters to convert the value of a feature attribute into a continuous-valued parameter such as color, size, width, opacity, etc.



The transformation is defined by a set of *(input, output)* control points chosen along a desired mapping curve. Piecewise interpolation along the curve is used to compute an output value for any input value.

The function is able to compute either numeric or color values as output. This is known as the **interpolation method**, and is specified by an optional parameter with a value of `numeric` (the default) or `color`.

The *shape* of the mapping curve between control points is specified by the **interpolation mode**, which is an optional parameter with values of `linear` (the default), `cubic`, or `cosine`.

Example

Interpolating over color ranges allows concise definition of continuously-varying colors for chloropleth (thematic) maps. As an example, consider a map of the US states dataset using the fill color to indicate the population of the states. The dataset has an attribute `PERSONS` containing the population of each state. The population values lie in the range 0 to around 30,000,000. The interpolation curve is defined by three control points which assign colors to the population levels 0, 9,000,000 and 23,000,000. The colors for population values are computed by piecewise linear interpolation along this curve. For example, a state with a population of 16,000,000 is displayed with a color midway between the ones for the middle and upper control points. States with populations greater than 23,000,000 are displayed with the last color.

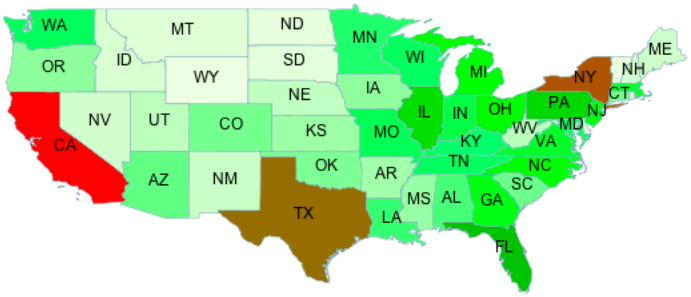
Because the interpolation is being performed over color values, the method parameter is supplied, with a value of `color`. Since the default linear interpolation is used, no interpolation mode is supplied,

The symbolizer for this style is:

```
<PolygonSymbolizer>
   <Fill>
     <CssParameter name="fill">
       <ogc:Function name="Interpolate">
         <!-- Property to transform -->
         <ogc:PropertyName>PERSONS</ogc:PropertyName>
```

```
                <!
↪-- Mapping curve definition pairs (input, output) -->
                <ogc:Literal>0</ogc:Literal>
                <ogc:Literal>#fefeee</ogc:Literal>

                <ogc:Literal>9000000</ogc:Literal>
                <ogc:Literal>#00ff00</ogc:Literal>

                <ogc:Literal>23000000</ogc:Literal>
                <ogc:Literal>#ff0000</ogc:Literal>

                <!-- Interpolation method -->
                <ogc:Literal>color</ogc:Literal>


↪               <!-- Interpolation mode - defaults to linear -->
              </ogc:Function>
            </CssParameter>
          </Fill>
        </PolygonSymbolizer>
```

This symbolizer produces the following output:



6.2.7 i18N in SLD

This section describes how to specify metadata (titles and abstracts) in different languages in SLD documents.

Metadata in different languages

GeoServer extends Title and Abstract sections, so that text in different languages can be included.

This is an example of the syntax to use:

```
<Title>This is the default title
  <Localized lang="en">English title</Localized>
  <Localized lang="it">Titolo in italiano</Localized>
</Title>
```

A default text (`This is the default title` in the example) and a set of Localized sections, one for each language that you want to support.

Each `Localized` section specifies the language (using a two letter abbreviation in the `lang` attribute) and the related text.

Currently, GeoServer supports localized text in SLD in WMS GetLegendGraphic requests (legends that contain labels are rendered using the requested language, if a `LANGUAGE` parameter is added to the request, e.g. `LANGUAGE=it`).

6.3 Generating SLD styles with QGIS

QGIS includes a sophisticated style editor with many map rendering possibilities. Styles generated with QGIS can then be exported (with limitations) to SLD for usage with GeoServer.

QGIS style exporting abilities have been evolving over time, as a reference:

- For vector data QGIS exports SLD 1.1 styles that can be read by GeoServer. In order to get the suitable results it's important to use QGIS 3.0 or newer, and GeoServer 2.13.x or newer.
- Raster data styling export is new in QGIS 3.4.5 (yet to be released at the time of writing). This new version exports SLD 1.0 styles with vendor extensions to support constrast streching that most recent GeoServer versions support properly. For older QGIS versions limited export functionality is available using the SLD4Raster plugin.

  For the export it is advised to use the *Save As* functionality available in the style dialog, as indicated below in this guide. Other plugins exist that streamline the export process, but they may ruin the style trying to adapt it to older GeoServer versions (e.g., translating it down to SLD 1.0 by simple text processing means), or rewrite it entirely.

Warning: Despite the progress in the last years, it is known that not all QGIS rendering options are supported by SLD and/or by GeoServer (e.g. shapeburst symbology), and that support for exporting some parts is simply missing (e.g.. expression based symbology is supported in SLD, but QGIS won't export it). If you are interested, both projects would welcome sponsoring to improve the situation.

6.3.1 Exporting vector symbology

This is a step by step guide to style a GeoServer demo layer, `sfdem`.

1. Open **`QGIS`** (minimum version 3.0)
2. Load the `states.shp` dataset from the GeoServer data directory, `<GEOSERVER_DATA_DIR>/data/shapefiles/states.shp`
3. Double click the layer to open the *Properties* dialog and switch to the *Symbology* page.
4. Choose a *Graduated* rendering, on the `PERSONS` column, and click on *Classify* button to generate *1.5* standard deviations, select the *spectral*

color ramp, switch mode to *Quantile* and finally and click on the "*Classify* button to generate a 5 classes map, as shown in figure.

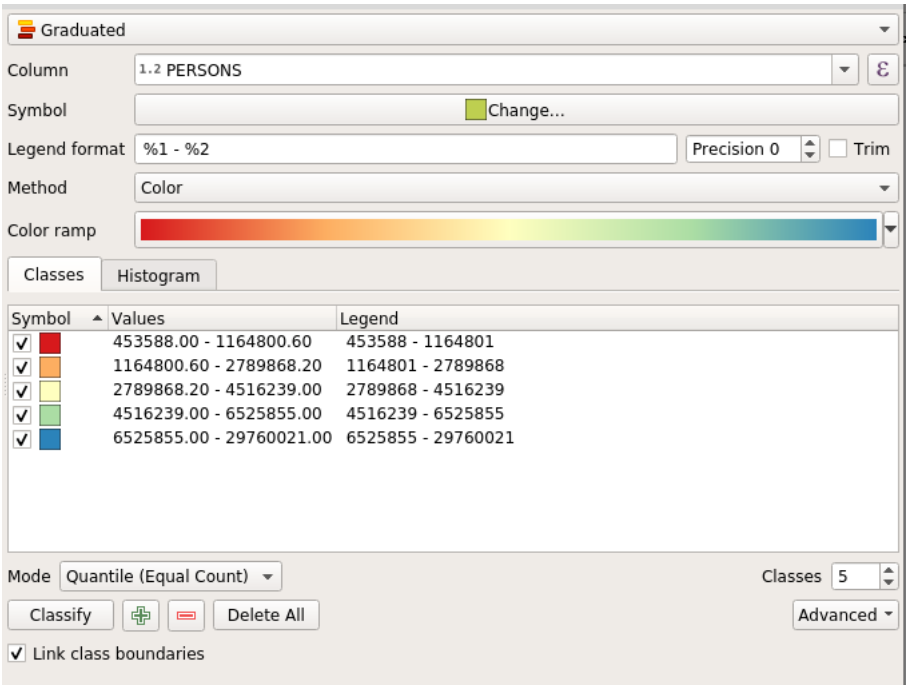


Fig. 6.89: QGIS vector styling

5. Switch to the *Labels* page, choose *Single labels'*, label with the `STATE NAME` attribute and choose your preferred text rendering options, as shown in figure
6. The layer renders as follows:
7. Go back At the *Properties* dialog, from the bottom of the *Styles* page, choose *Style* → *Save Style*.
8. Choose export in the SLD format, placing the file in the desired location.
9. Go in GeoServer, create a new style, use the *Upload a new style* dialog to choose the exported file, and click on *upload* link.
10. Click on guilabel:*Apply*.
11. Change to the *Layer preview* tab, click on the *Preview on Layer* link to choose `topp:states` to verify proper rendering.
12. Eventually switch to the *Publishing* tab, search for `states`, and select *Default* or *Associated* checkbox to publish the layer to use the new style permanently.

6.3.2 Exporting raster symbology

This is a step by step guide to style a GeoServer demo layer, `sfdem`.

1. Open QGIS (minimum version 3.4.5)

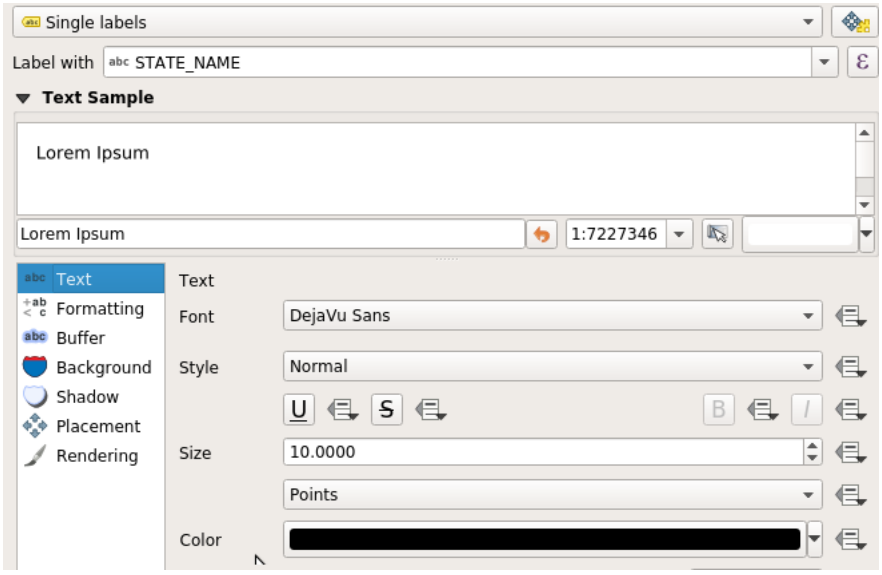


Fig. 6.90: QGIS labelling

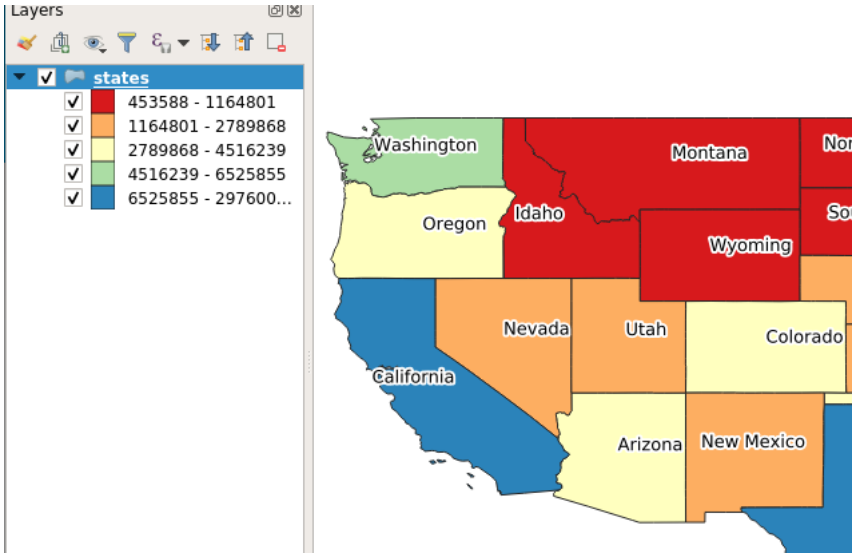


Fig. 6.91: QGIS raster styling

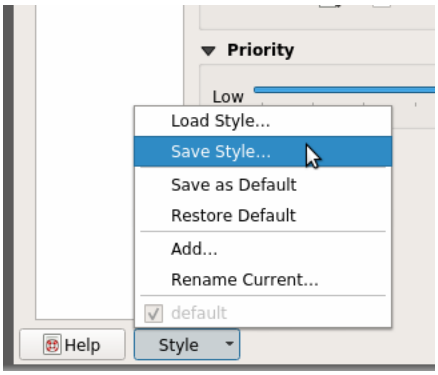


Fig. 6.92: *Export using Save As...*

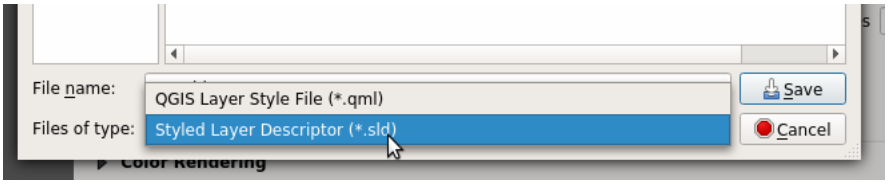


Fig. 6.93: Choosing export format...

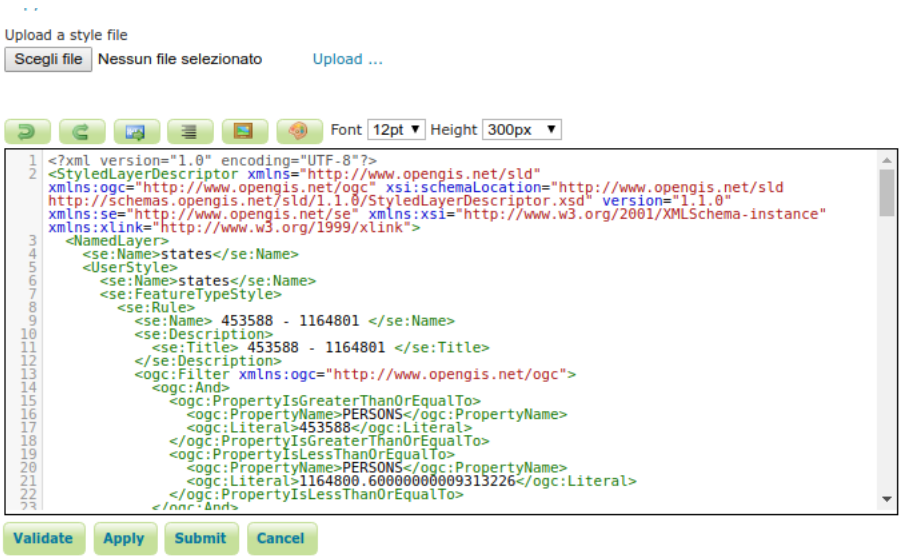


Fig. 6.94: Uploading style in GeoServer...

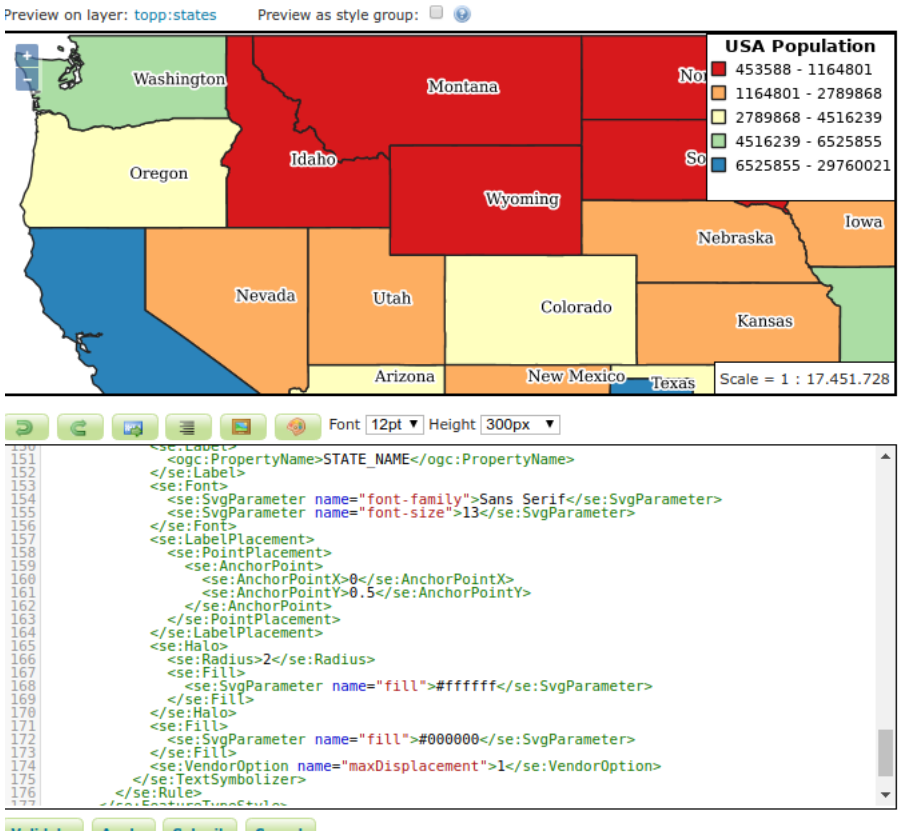


Fig. 6.95: Previewing style in GeoServer...

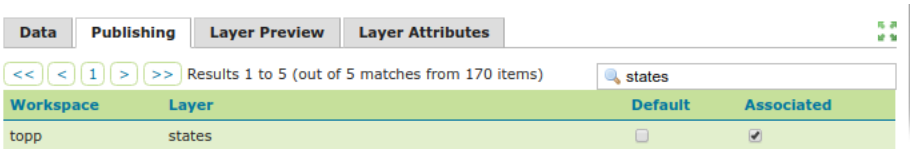


Fig. 6.96: Associating style in GeoServer...

2. Load the `sfdem.tif` raster from the GeoServer data directory, `<GEOSERVER_DATA_DIR>/data/sf/sfdem.tif`

3. Double click the layer to open the *Properties* dialog and switch to the *Symbology* page.

4. Choose a *Singleband pseudocolor* rendering, Generate *Min / Max Value Settings* using *Mean +/- standard deviation* with using `1.5` standard deviations. Generate a 5 classes *Linear* interpolated map, as shown in figure.

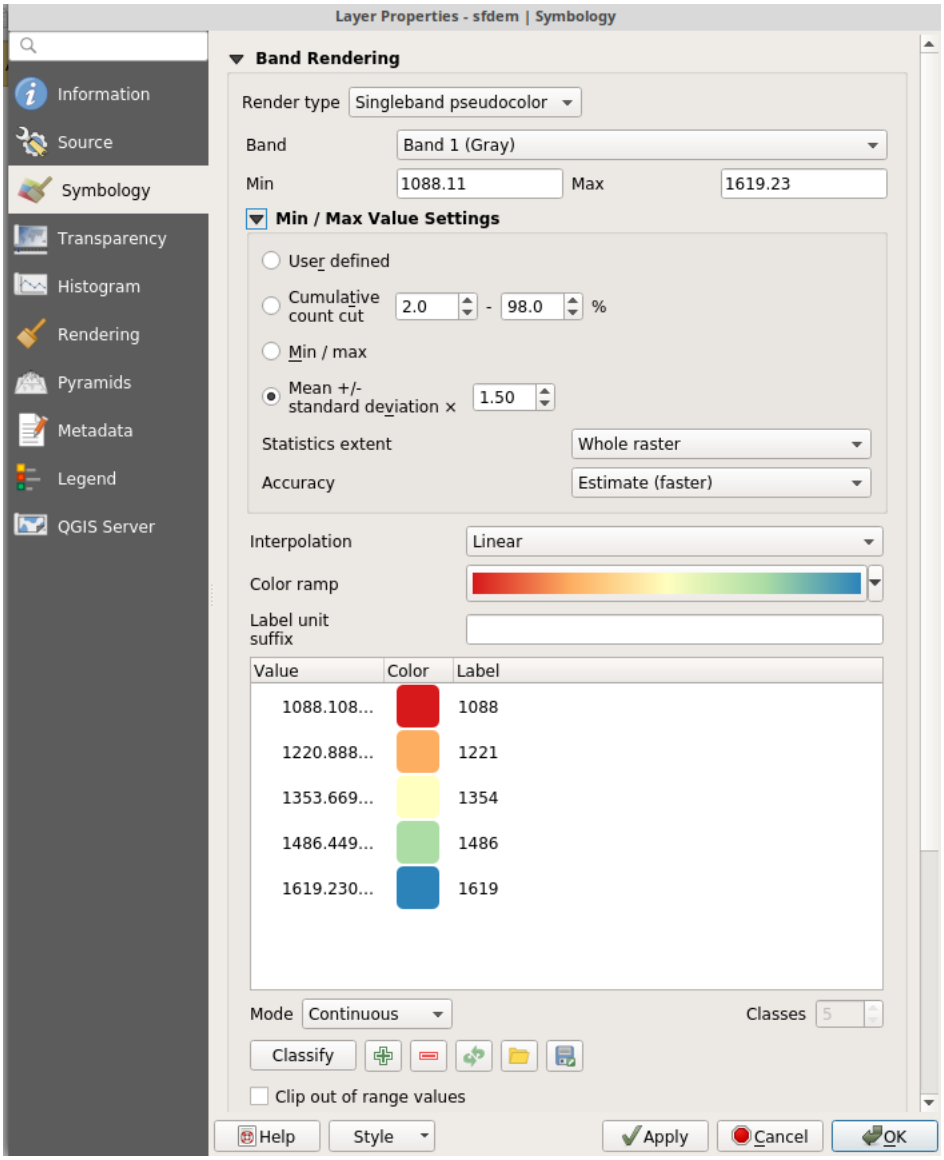


Fig. 6.97: QGIS raster styling

5. The layer renders as follows:

6. Return to the layer's *Properties* dialog *Symbology* page, at the bottom of the page choose *Style* → *Save Style*.

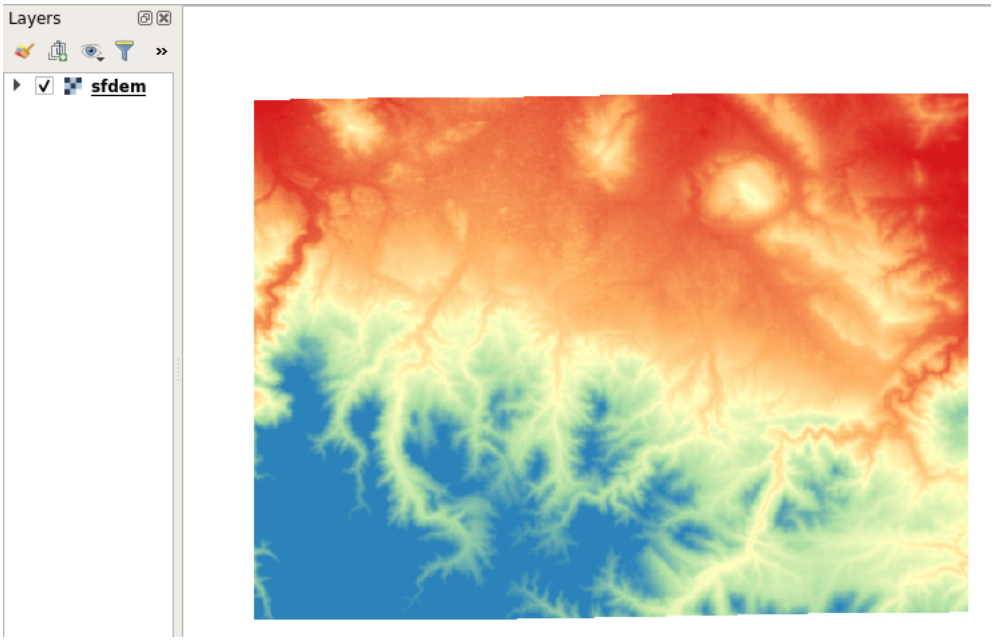


Fig. 6.98: QGIS raster styling

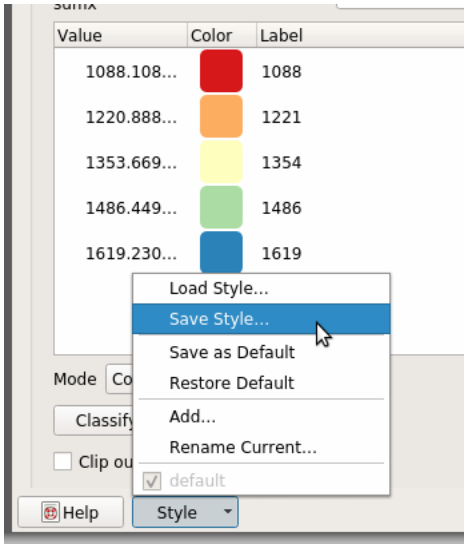


Fig. 6.99: Export using Save As...

7. Choose export in the SLD format, placing the file in the desired location

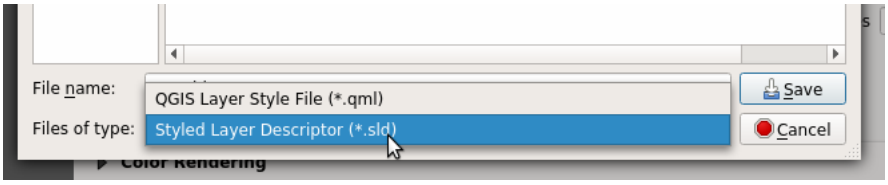


Fig. 6.100: Choosing export format...

8. Go in GeoServer, create a new style, use the *Upload a new style* dialog to choose the exported file, and click on *upload* link.

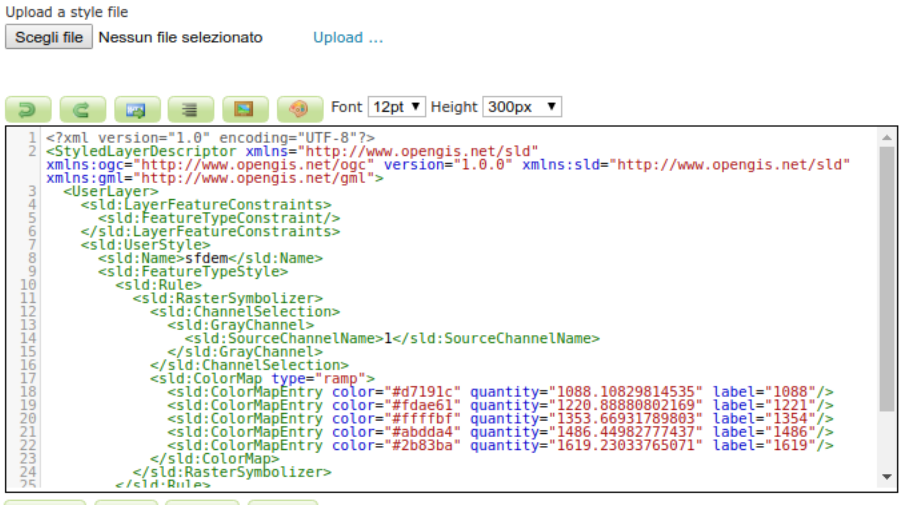


Fig. 6.101: Uploading style in GeoServer...

9. Click on guilabel:*Apply* then change to the *Layer preview* tab. Click on the *Preview on Layer* link to choose `sfdem` to verify proper rendering.

10. Finally switch to the *Publishing* tab, search for `sfdem` layer, and select *Default* or *Associated* checkbox to publish `sfdem` with the new style.

6.4 CSS Styling

The CSS extension uses a CSS-derived language instead of SLD. These CSS styles are internally converted to SLD, which is then used as normal by GeoServer. The CSS syntax is duplicated from SVG styling where appropriate, but extended to avoid losing facilities provided by SLD when possible.

CSS is not a part of GeoServer by default, but is available as an extension.

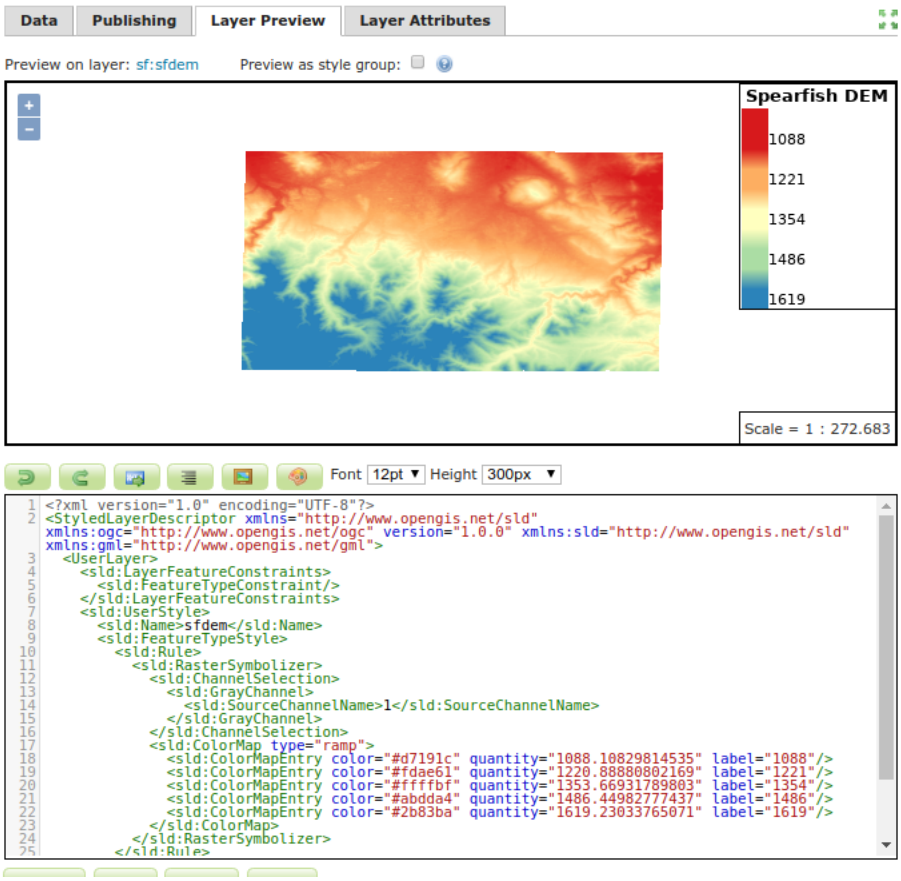


Fig. 6.102: Previewing style in GeoServer...

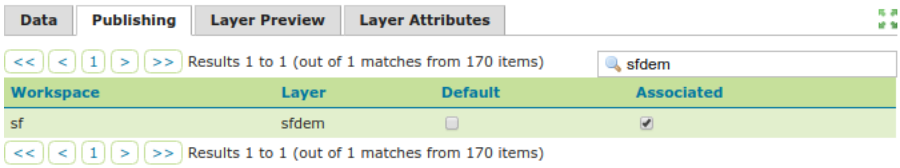


Fig. 6.103: Associating style in GeoServer...

6.4.1 Installation

The CSS extension is listed among the other extension downloads on the GeoServer download page.

The installation process is similar to other GeoServer extensions:

1. Download the appropriate archive from the GeoServer download page. Please verify that the version number in the filename corresponds to the version of GeoServer you are running. The file will be called `geoserver-A.B.C-css-plugin.zip` where `A.B.C` is the GeoServer version.
2. Extract the contents of the archive into the `WEB-INF/lib` directory in GeoServer. Make sure you do not create any sub-directories during the extraction process.
3. Restart GeoServer.

   If installation was successful, you will see a new CSS entry in the *Styles* editor.

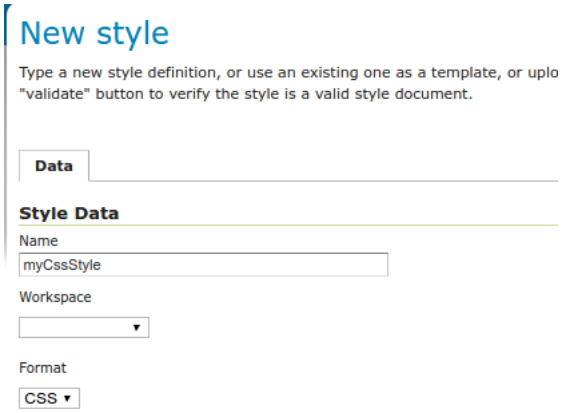


Fig. 6.104: CSS format in the new style page

After installation, you may wish to read the tutorial: *Styling data with CSS*.

6.4.2 Tutorial: Styling data with CSS

This tutorial will show using CSS to style a layer, along with the equivalent SLD code.

To use this tutorial, you will need the *CSS extension* as well as the `states` layer from the default GeoServer configuration.

Creating a style for the states layer

The SLD file for the default `states` layer looks like this:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<StyledLayerDescriptor
  version="1.0.0"
  xmlns="http://www.opengis.net/sld"
  xmlns:ogc="http://www.opengis.net/ogc"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:gml="http://www.opengis.net/gml"
  xsi:schemaLocation="http://www.opengis.net/sld
    http://schemas.
↪opengis.net/sld/1.0.0/StyledLayerDescriptor.xsd
">
  <NamedLayer>
    <Name>USA states population</Name>
    <UserStyle>
      <Name>population</Name>
      <Title>Population in the United States</Title>
      <Abstract>A sample␣
↪filter that filters the United States into three
        categories␣
↪of population, drawn in different colors</Abstract>
      <FeatureTypeStyle>
        <Rule>
          <Title>&lt; 2M</Title>
          <ogc:Filter>
            <ogc:PropertyIsLessThan>
␣
↪          <ogc:PropertyName>PERSONS</ogc:PropertyName>
              <ogc:Literal>2000000</ogc:Literal>
            </ogc:PropertyIsLessThan>
          </ogc:Filter>
          <PolygonSymbolizer>
            <Fill>
              <!-- CssParameters␣
↪allowed are fill (the color) and fill-opacity -->
␣
↪       <CssParameter name="fill">#4DFF4D</CssParameter>
␣
↪ <CssParameter name="fill-opacity">0.7</CssParameter>
            </Fill>
          </PolygonSymbolizer>
        </Rule>
        <Rule>
          <Title>2M - 4M</Title>
          <ogc:Filter>
            <ogc:PropertyIsBetween>
␣
↪          <ogc:PropertyName>PERSONS</ogc:PropertyName>
              <ogc:LowerBoundary>
                <ogc:Literal>2000000</ogc:Literal>
              </ogc:LowerBoundary>
              <ogc:UpperBoundary>
                <ogc:Literal>4000000</ogc:Literal>
              </ogc:UpperBoundary>
            </ogc:PropertyIsBetween>
          </ogc:Filter>
          <PolygonSymbolizer>
            <Fill>
```

```
                    <!-- CssParameters␣
↪allowed are fill (the color) and fill-opacity -->
␣
↪     <CssParameter name="fill">#FF4D4D</CssParameter>
                  ␣
↪ <CssParameter name="fill-opacity">0.7</CssParameter>
                </Fill>
              </PolygonSymbolizer>
            </Rule>
            <Rule>
              <Title>&gt; 4M</Title>
              ␣
↪   <!-- like a linesymbolizer but with a fill too -->
              <ogc:Filter>
                <ogc:PropertyIsGreaterThan>
                ␣
↪          <ogc:PropertyName>PERSONS</ogc:PropertyName>
                  <ogc:Literal>4000000</ogc:Literal>
                </ogc:PropertyIsGreaterThan>
              </ogc:Filter>
              <PolygonSymbolizer>
                 <Fill>
                    <!-- CssParameters␣
↪allowed are fill (the color) and fill-opacity -->
                 ␣
↪     <CssParameter name="fill">#4D4DFF</CssParameter>
                  ␣
↪ <CssParameter name="fill-opacity">0.7</CssParameter>
                </Fill>
              </PolygonSymbolizer>
            </Rule>
            <Rule>
              <Title>Boundary</Title>
              <LineSymbolizer>
                <Stroke>
                ␣
↪ <CssParameter name="stroke-width">0.2</CssParameter>
                </Stroke>
              </LineSymbolizer>
              <TextSymbolizer>
                <Label>
                ␣
↪      <ogc:PropertyName>STATE_ABBR</ogc:PropertyName>
                </Label>
                <Font>
                  <CssParameter␣
↪name="font-family">Times New Roman</CssParameter>
                ␣
↪<CssParameter name="font-style">Normal</CssParameter>
                ␣
↪     <CssParameter name="font-size">14</CssParameter>
                </Font>
                <LabelPlacement>
                  <PointPlacement>
                    <AnchorPoint>
                      <AnchorPointX>0.5</AnchorPointX>
                      <AnchorPointY>0.5</AnchorPointY>
                    </AnchorPoint>
```

```
                </PointPlacement>
              </LabelPlacement>
            </TextSymbolizer>
          </Rule>
        </FeatureTypeStyle>
      </UserStyle>
      </NamedLayer>
    </StyledLayerDescriptor>
```

Now, let's start on a CSS file that accomplishes the same thing.

First, got to the styles page and click on *Add a new style* link to start a new style. In the "New style" page, do the following:

- Name the new style anything you'd like, such as `csstutorial`
- Choose `CSS` as the format
- In the *Generate a default style* dropdown choose `Polygon` and click on *Generate...*

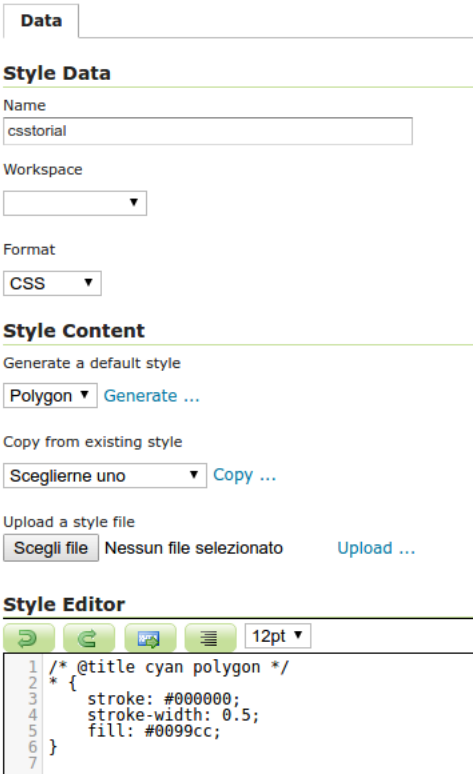


Fig. 6.105: Creating a new CSS style

This creates an example style with a source similar to this one (the colors may differ):

```
/* @title cyan polygon */
* {
    stroke: #000000;
    stroke-width: 0.5;
```

```
    fill: #0099cc;
}
```

This demonstrates the basic elements of a CSS style:

A **selector** that identifies some part of the data to style. Here, the selector is `*`, indicating that all data should use the style properties.

Properties inside curly braces (`{}`) which specify how the affected features should be styled. Properties consist of name/value pairs separated by colons (`:`).

We can also see the basics for styling a polygon (`fill`), and its outline (`stroke`).

See also:

The *Filter syntax* and *Property listing* pages provide more information about the options available in CSS styles.

Before moving on, let's save the style and preview it with the states layer:

- Click on "Apply" to save the layer and enable the style preview
- Now on the "Style Editor page", switch to the "layer preview" tab and click on the "previewing on layer" link, then choose the "states" layer in the dialog
- The style editor should now show the states layer filled and stroked

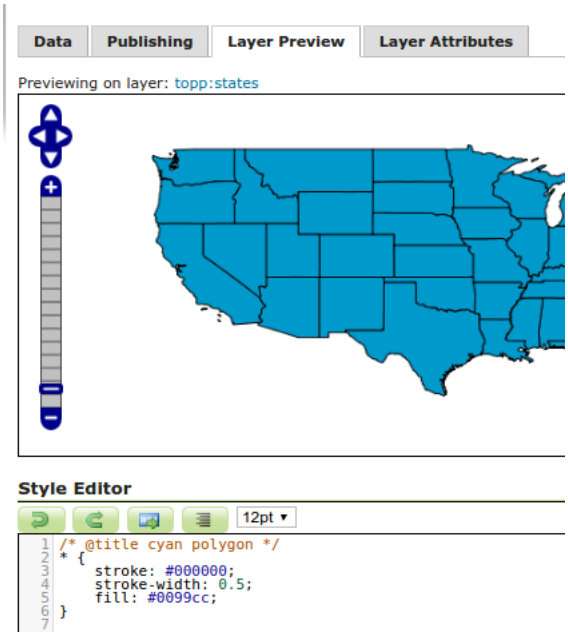


Fig. 6.106: Previewing the CSS style with the state layer

Let's use these basics to start translating the states style. The first rule in the SLD applies to states where the `PERSONS` field is less than two million:

```
<Rule>
  <Title>&lt; 2M</Title>
  <ogc:Filter>
    <ogc:PropertyIsLessThan>
     <ogc:PropertyName>PERSONS</ogc:PropertyName>
     <ogc:Literal>2000000</ogc:Literal>
    </ogc:PropertyIsLessThan>
  </ogc:Filter>
  <PolygonSymbolizer>
     <Fill>
        <!-- CssParameters␣
↪allowed are fill (the color) and fill-opacity -->
  ␣
↪      <CssParameter name="fill">#4DFF4D</CssParameter>
      ␣
↪  <CssParameter name="fill-opacity">0.7</CssParameter>
     </Fill>
  </PolygonSymbolizer>
</Rule>
```

Using a *CQL*-based selector, and copying the names and values of the CssParameters over, we get:

```
[PERSONS < 2000000] {
  fill: #4DFF4D;
  fill-opacity: 0.7;
}
```

For the second style, we have a `PropertyIsBetween` filter, which doesn't directly translate to CSS:

```
<Rule>
  <Title>2M - 4M</Title>
  <ogc:Filter>
    <ogc:PropertyIsBetween>
      <ogc:PropertyName>PERSONS</ogc:PropertyName>
      <ogc:LowerBoundary>
        <ogc:Literal>2000000</ogc:Literal>
      </ogc:LowerBoundary>
      <ogc:UpperBoundary>
        <ogc:Literal>4000000</ogc:Literal>
      </ogc:UpperBoundary>
    </ogc:PropertyIsBetween>
  </ogc:Filter>
  <PolygonSymbolizer>
     <Fill>
        <!-- CssParameters␣
↪allowed are fill (the color) and fill-opacity -->
  ␣
↪      <CssParameter name="fill">#FF4D4D</CssParameter>
      ␣
↪  <CssParameter name="fill-opacity">0.7</CssParameter>
     </Fill>
  </PolygonSymbolizer>
</Rule>
```

However, `PropertyIsBetween` can easily be replaced by a combination of two comparison selectors. In CSS, you can apply multiple

selectors to a rule by simply placing them one after the other. Selectors separated by only whitespace must all be satisfied for a style to apply. Multiple such groups can be attached to a rule by separating them with commas (`,`). If a feature matches any of the comma-separated groups for a rule then that style is applied. Thus, the CSS equivalent of the second rule is:

```
[PERSONS >= 2000000] [PERSONS < 4000000] {
  fill: #FF4D4D;
  fill-opacity: 0.7;
}
```

The third rule can be handled in much the same manner as the first:

```
[PERSONS >= 4000000] {
  fill: #4D4DFF;
  fill-opacity: 0.7;
}
```

The fourth and final rule is a bit different. It applies a label and outline to all the states:

```
<Rule>
  <Title>Boundary</Title>
  <LineSymbolizer>
    <Stroke>
    <CssParameter name="stroke-width">0.2</CssParameter>
    </Stroke>
  </LineSymbolizer>
  <TextSymbolizer>
    <Label>
      <ogc:PropertyName>STATE_ABBR</ogc:PropertyName>
    </Label>
    <Font>
      <CssParameter name="font-family">Times New Roman</CssParameter>
    <CssParameter name="font-style">Normal</CssParameter>
      <CssParameter name="font-size">14</CssParameter>
    </Font>
    <LabelPlacement>
      <PointPlacement>
        <AnchorPoint>
          <AnchorPointX>0.5</AnchorPointX>
          <AnchorPointY>0.5</AnchorPointY>
        </AnchorPoint>
      </PointPlacement>
    </LabelPlacement>
  </TextSymbolizer>
</Rule>
```

This introduces the idea of rendering an extracted value (`STATE_ABBR`) directly into the map, unlike all of the rules thus far. For this, you can use a CQL expression wrapped in square braces (`[]`) as the value of a CSS property. It is also necessary to surround values containing whitespace, such as `Times New Roman`, with single- or double-quotes (`"`, `'`). With these details in mind,

let's write the rule:

```
* {
  stroke-width: 0.2;
  label: [STATE_ABBR];
  label-anchor: 0.5 0.5;
  font-family: "Times New Roman";
  font-fill: black;
  font-style: normal;
  font-size: 14;
}
```

Putting it all together, you should now have a style that looks like:

```
[PERSONS < 2000000] {
  fill: #4DFF4D;
  fill-opacity: 0.7;
}

[PERSONS >= 2000000] [PERSONS < 4000000] {
  fill: #FF4D4D;
  fill-opacity: 0.7;
}

[PERSONS >= 4000000] {
  fill: #4D4DFF;
  fill-opacity: 0.7;
}

* {
  stroke-width: 0.2;
  label: [STATE_ABBR];
  label-anchor: 0.5 0.5;
  font-family: "Times New Roman";
  font-fill: black;
  font-style: normal;
  font-size: 14;
}
```

Click the *Apply* button at the bottom of the form to save your changes.

You will see that the borders are missing! In the GeoServer CSS module, each type of symbolizer has a "key" property which controls whether it is applied. Without these "key" properties, subordinate properties are ignored. These "key" properties are:

- **fill**, which controls whether or not Polygon fills are applied. This specified the color or graphic to use for the fill.
- **stroke**, which controls whether or not Line and Polygon outline strokes are applied. This specifies the color (or graphic fill) of the stroke.
- **mark**, which controls whether or not point markers are drawn. This identifies a Well-Known Mark or image URL to use.
- **label**, which controls whether or not to draw labels on the map. This identifies the text to use for labeling the map, usually as a CQL expression.

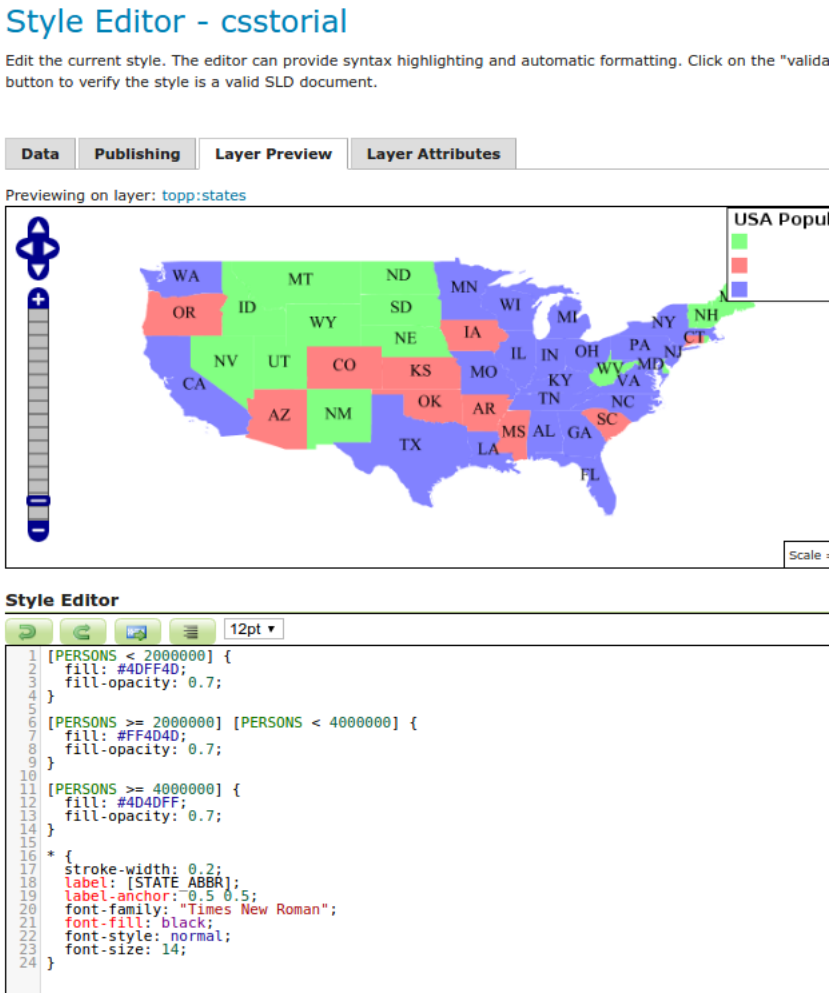


Fig. 6.107: CSS style applied to the `states` layer

- **halo-radius**, which controls whether or not to draw a halo around labels. This specifies how large such halos should be.

See also:

The *Property listing* page for information about the other properties.

Since we don't specify a `stroke` color, no stroke is applied. Let's add it, replacing the final rule so that it will now look like this:

```
* {
  stroke: black;
  stroke-width: 0.2;
  label: [STATE_ABBR];
  label-anchor: 0.5 0.5;
  font-family: "Times New Roman";
  font-fill: black;
  font-style: normal;
  font-size: 14;
}
```

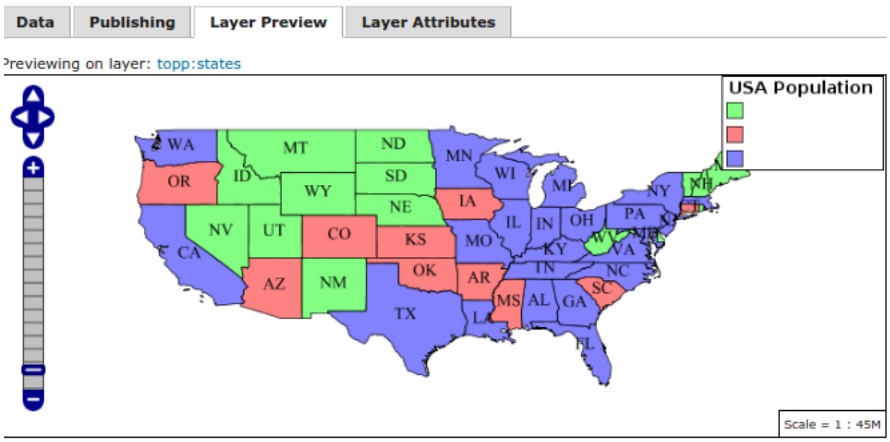


Fig. 6.108: Border added to style

Refining the style

Removing duplicated properties

The style that we have right now is only 23 lines, a nice improvement over the 103 lines of XML that we started with. However, we are still repeating the `fill-opacity` attribute everywhere.

We can move it into the `*` rule and have it applied everywhere. This works because the GeoServer CSS module emulates *cascading*: While SLD uses a "painter's model" where each rule is processed independently, a cascading style allows you to provide general style properties and override only specific properties for particular features.

This brings the style down to only 21 lines:

```
[PERSONS < 2000000] {
  fill: #4DFF4D;
}

[PERSONS > 2000000] [PERSONS < 4000000] {
  fill: #FF4D4D;
}

[PERSONS > 4000000] {
  fill: #4D4DFF;
}

* {
  fill-opacity: 0.7;
  stroke-width: 0.2;
  label: [STATE_ABBR];
  label-anchor: 0.5 0.5;
  font-family: "Times New Roman";
  font-fill: black;
  font-style: normal;
  font-size: 14;
}
```

Scale-dependent styles

The labels for this style are nice, but at lower zoom levels they seem a little crowded. We can easily move the labels to a rule that doesn't activate until the scale denominator is below 2000000. We do want to keep the stroke and fill-opacity at all zoom levels, so we can separate them from the label properties.

Keep the following properties in the main (*) rule:

```
* {
  fill-opacity: 0.7;
  stroke-width: 0.2;
}
```

Remove all the rest, moving them into a new rule:

```
[@sd < 20M] {
  label: [STATE_ABBR];
  label-anchor: 0.5 0.5;
  font-family: "Times New Roman";
  font-fill: black;
  font-style: normal;
  font-size: 14;
}
```

Setting titles for the legend

So far, we haven't set titles for any of the style rules. This doesn't really cause any problems while viewing maps, but GeoServer uses the title in auto-generating legend graphics. Without the titles,

GeoServer falls back on the names, which in the CSS module are generated from the filters for each rule. Titles are not normally a part of CSS, so GeoServer looks for them in specially formatted comments before each rule. We can add titles like this:

```
/* @title Population < 2M */
[PERSONS < 2000000] {

  ...

/* @title 2M < Population < 4M */
[PERSONS > 2000000] [PERSONS < 4000000] {

  ...

/* @title Population > 4M */
[PERSONS > 4000000] {

  ...


/* @title Boundaries */
* {

  ...
```

Because of the way that CSS is translated to SLD, each SLD rule is a combination of several CSS rules. This is handled by combining the titles with the word "with". If the title is omitted for a rule, then it is simply not included in the SLD output.

The final CSS should looks like this:

```
/* @title Population < 2M */
[PERSONS < 2000000] {
  fill: #4DFF4D;
  fill-opacity: 0.7;
}

/* @title 2M < Population < 4M */
[PERSONS >= 2000000] [PERSONS < 4000000] {
  fill: #FF4D4D;
  fill-opacity: 0.7;
}

/* @title Population > 4M */
[PERSONS >= 4000000] {
  fill: #4D4DFF;
  fill-opacity: 0.7;
}


/* @title Boundaries */
* {
  stroke: black;
  stroke-width: 0.2;
  fill-opacity: 0.7;
}
```

```
[@sd < 20M] {
  label: [STATE_ABBR];
  label-anchor: 0.5 0.5;
  font-family: "Times New Roman";
  font-fill: black;
  font-style: normal;
  font-size: 14;
}
```

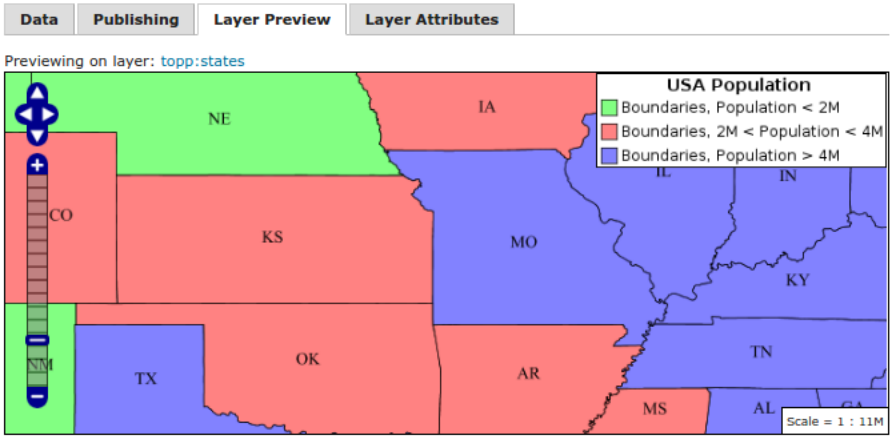


Fig. 6.109: Final style with rule names

Applying rule nesting

As a final variation, the style can be made more compact by leveraging rule nesting:

```
* {
  stroke: black;
  stroke-width: 0.2;
  fill-opacity: 0.7;

  /* @title Population < 2M */
  [PERSONS < 2000000] {
    fill: #4DFF4D;
  };
  /* @title 2M < Population < 4M */
  [PERSONS >= 2000000] [PERSONS < 4000000] {
    fill: #FF4D4D;
  };
  /* @title Population > 4M */
  [PERSONS >= 4000000] {
    fill: #4D4DFF;
  };

  /* Labelling */
  [@sd < 20M] {
    label: [STATE_ABBR];
    label-anchor: 0.5 0.5;
```

```
    font-family: "Times New Roman";
    font-fill: black;
    font-style: normal;
    font-size: 14;
  }
}
```

CSS Workshop

For more details, visit the next section, the CSS workshop. This workshop has been used in the past for classroom settings to teach the CSS extension and has been ported to the user documentation.

6.4.3 Filter syntax

Filters limit the set of features affected by a rule's properties. There are several types of simple filters, which can be combined to provide more complex filters for rules.

Combining filters

Combination is done in the usual CSS way. A rule with two filters separated by a comma affects any features that match *either* filter, while a rule with two filters separated by only whitespace affects only features that match *both* filters. Here's an example using a basic attribute filter (described below):

```
/* Matches places where the lake is flooding */
[rainfall>12] [lakes>1] {
    fill: black;
}

/* Matches wet places */
[rainfall>12], [lakes>1] {
    fill: blue;
}
```

When writing a selector that uses both *and* and *or* combinators, remember that the *and* combinator has higher precedence. For example:

```
restricted [cat='2
↪'], [cat='3'], [cat='4'] [@sd <= 200k] [@sd > 100k] {
  fill: #EE0000;
}
```

The above selector should be read as:

- typename is 'restricted' and `cat='2'` *or*
- `cat='3'` *or*
- `cat='4'` and scale is between 100000 and 200000

If instead the intention was to combine in or just the three cat filters, the right syntax would have been:

```
restricted [cat='2
↪' or cat='3' or cat='4'] [@sd <= 200k] [@sd > 100k] {
  fill: #EE0000;
}
```

Which should be read as:

- typename is 'restricted' *and*
- (`cat='2'` or `cat='3'` or `cat='4'`) *and*
- scale is between 100000 and 200000

Filtering on data attributes

An attribute filter matches some attribute of the data (for example, a column in a database table). This is probably the most common type of filter. An attribute filter takes the form of an attribute name and a data value separated by some predicate operator (such as the less-than operator <).

Supported predicate operators include the following:

Operator	Meaning
=	The property must be exactly *equal* to the specified value.
<>	The property must not be exactly equal to the specified value.
>	The property must be greater than (or alphabetically later than) the specified value.
>=	The property must be greater than or equal to the specified value.
<	The property must be less than (or alphabetically earlier than) the specified value.
<=	The property must be less than or equal to the specified value.
LIKE	The property must match the pattern described by the specified value. Patterns use _ to indicate a single unspecified character and % to indicate an unknown number of unspecified characters.

For example, to only render outlines for the states whose names start with letters in the first half of the alphabet, the rule would look like:

```
[STATE_NAME<='M'] {
    stroke: black;
}
```

Note: The current implementation of property filters uses ECQL syntax, described on the GeoTools documentation.

Filtering on type

When dealing with data from multiple sources, it may be useful to provide rules that only affect one of those sources. This is done very simply; just specify the name of the layer as a filter:

```
states {
    stroke: black;
}
```

Filtering by ID

For layers that provide feature-level identifiers, you can style specific features simply by specifying the ID. This is done by prefixing the ID with a hash sign (#):

```
#states.2 {
    stroke: black;
}
```

Note: In CSS, the `.` character is not allowed in element ids; and the `#states.foo` selector matches the element with id `states` only if it also has the class `foo`. Since this form of identifier comes up so frequently in GeoServer layers, the CSS module deviates from standard CSS slightly in this regard. Future revisions may use some form of munging to avoid this deviation.

Filtering by rendering context (scale)

Often, there are aspects of a map that should change based on the context in which it is being viewed. For example, a road map might omit residential roads when being viewed at the state level, but feature them prominently at the neighborhood level. Details such as scale level are presented as pseudo-attributes; they look like property filters, but the property names start with an `@` symbol:

```
[roadtype = 'Residential'][@sd > 100k] {
    stroke: black;
}
```

The context details that are provided are as follows:

Pseudo-Attribute	Meaning
@sd	The scale denominator for the current rendering. More explicitly, this is the ratio of real-world distance to screen/rendered distance.
@scale	Same as above, the scale denominator (not scale) for the current rendering. Supported for backwards compatibility

The scale value can be expressed as a plain number, for for brevity and readability the suffixes k (kilo), M (mega), G (giga) can be used, for example:

```
[@sd > 100k]
[@sd < 12M]
[@sd < 1G]
```

Note: While property filters (currently) use the more complex ECQL syntax, pseudo-attributes cannot use complex expressions and MUST take the form of <PROPERTY><OPERATOR><LITERAL>.

Filtering symbols

When using symbols to create graphics inline, you may want to apply some styling options to them. You can specify style attributes for built-in symbols by using a few special selectors:

PseudoSelector	Meaning
`:mark`	specifies that a rule applies to symbols used as point markers
`:stroke`	specifies that a rule applies to symbols used as stroke patterns
`:fill`	specifies that a rule applies to symbols used as fill patterns
`:symbol`	specifies that a rule applies to any symbol, regardless of which context it is used in
`:nth-mark(n)`	specifies that a rule applies to the symbol used for the nth stacked point marker on a feature.
`:nth-stroke(n)`	specifies that a rule applies to the symbol used for the nth stacked stroke pattern on a feature.
`:nth-fill(n)`	specifies that a rule applies to the symbol used for the nth stacked fill pattern on a feature.
`:nth-symbol(n)`	specifies that a rule applies to the symbol used for the nth stacked symbol on a feature, regardless of which context it is used in.

For more discussion on using these selectors, see *Styled marks*.

Global rules

Sometimes it is useful to have a rule that matches all features, for example, to provide some default styling for your map (remember, by default nothing is rendered). This is accomplished using a single asterisk * in place of the usual filter. This catch-all rule can be used in complex expressions, which may be useful if you want a rule to provide defaults as well as overriding values for some features:

```
* {
    stroke: black;
}
```

6.4.4 Metadata

One feature that appears in SLD that has no analog in CSS is the ability to provide *metadata* for styles and style rules. For example, this SLD embeds a title for its single rule:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<StyledLayerDescriptor version="1.0.0"
    xmlns="http://www.opengis.net/sld"
    xmlns:ogc="http://www.opengis.net/ogc"
```

```
    xmlns:xlink="http://www.w3.org/1999/xlink"
    ␣
↪xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:gml="http://www.opengis.net/gml"
    xsi:schemaLocation="http://www.opengis.net/sld
        http://schemas.
↪opengis.net/sld/1.0.0/StyledLayerDescriptor.xsd"
>
  <NamedLayer>
    <Name>Country Borders</Name>
    <UserStyle>
      <Name>borders</Name>
      <Title>Country Borders</Title>
      <Abstract>
          Borders of␣
↪countries, in an appropriately sovereign aesthetic.
      </Abstract>
      <FeatureTypeStyle>
        <Rule>
          <Title>Borders</Title>
          <LineSymbolizer>
            <Stroke>
            ␣
↪ <CssParameter name="stroke-width">0.2</CssParameter>
            </Stroke>
          </LineSymbolizer>
        </Rule>
     </FeatureTypeStyle>
    </UserStyle>
  </NamedLayer>
</StyledLayerDescriptor>
```

Software such as GeoServer can use this metadata to automatically generate nice legend images directly from the style. You don't have to give up this ability when styling maps in CSS; just add comment *before* your rules including lines that start with `@title` and `@abstract`. Here is the analogous style in CSS:

```
/*
 * @title This is a point layer.
 * @abstract This is an abstract point layer.
 */
* {
    mark: mark(circle);
}
```

Rules can provide either a title, an abstract, both, or neither. The SLD Name for a rule is autogenerated based on the filters from the CSS rules that combined to form it, for aid in troubleshooting.

Combined rules

One thing to keep in mind when dealing with CSS styles is that multiple rules may apply to the same subset of map features, especially as styles get more complicated. Metadata is inherited similarly to CSS properties, but metadata fields are **combined** instead of over-

riding less specific rules. That means that when you have a style like this:

```
/* @title Borders */
* {
    stroke: black;
}

/* @title Parcels */
[category='parcel'] {
    fill: blue;
}
```

The legend entry for parcels will have the title `'Parcels with Borders'`. If you don't like this behavior, then only provide titles for the most specific rules in your style. (Or, suggest something better in an issue report!) Rules that don't provide titles are simply omitted from title aggregation.

6.4.5 Multi-valued properties

When rendering maps, it is sometimes useful to draw the same feature multiple times. For example, you might want to stroke a roads layer with a thick line and then a slimmer line of a different color to create a halo effect.

In GeoServer's `css` module, all properties may have multiple values. There is a distinction between complex properties, and multi-valued properties. Complex properties are separated by spaces, while multi-valued properties are separated by commas. So, this style fills a polygon once:

```
* {
    fill: url("path/to/img.png") red;
}
```

Using `red` as a fallback color if the image cannot be loaded. If you wanted to draw red on top of the image, you would have to style like so:

```
* {
    fill: url("path/to/img.png"), red;
    /* set a transparency for the second fill,
       leave the first fully opaque. */
    fill-opacity: 100%, 20%;
}
```

For each type of symbolizer (`fill`, `mark`, `stroke`, and `label`) the number of values determines the number of times the feature will be drawn. For example, you could create a bulls-eye effect by drawing multiple circles on top of each other with decreasing sizes:

```
* {
    mark: symbol(circle),
↪ symbol(circle), symbol(circle), symbol(circle);
    mark-size: 40px, 30px, 20px, 10px;
```

```
}
```

If you do not provide the same number of values for an auxiliary property, the list will be repeated as many times as needed to finish. So:

```
* {
    mark: symbol(circle),
↪ symbol(circle), symbol(circle), symbol(circle);
    mark-size: 40px, 30px, 20px, 10px;
    mark-opacity: 12%;
}
```

makes all those circles 12% opaque. (Note that they are all drawn on top of each other, so the center one will appear 4 times as solid as the outermost one.)

Inheritance

For purposes of inheritance/cascading, property lists are treated as indivisible units. For example:

```
* {
    stroke: red, green, blue;
    stroke-width: 10px, 6px, 2px;
}

[type='special'] {
    stroke: pink;
}
```

This style will draw the 'special' features with only one outline. It has `stroke-width: 10px, 6px, 2px;` so that outline will be 10px wide.

6.4.6 Property listing

This page lists the supported rendering properties. See *CSS value types* for more information about the value types for each.

Point symbology

Property	Type	Meaning	Accepts Expression?
`mark`	url, symbol	The image or well-known shape to render for points	yes
`mark-composite`	string	The composite mode to be used and the optional opacity separated with a comma. See the *full list of available modes*.	no
`mark-mime`	string (MIME Type)	The type of the image referenced by a url()	No, defaults to 'image/jpeg'
`mark-geometry`	expression	An expression to use for the geometry when rendering features	yes
`mark-size`	length	The width to assume for the provided image. The height will be adjusted to preserve the source aspect ratio.	yes
`mark-rotation`	angle	A rotation to be applied (clockwise) to the mark image.	yes
`z-index`	integer	Controls the z ordering of output	no
`mark-label-obstacle`	boolean	If true the point symbol will be consider an obstable for labels, no label will overlap it	no

Line symbology

Property	Type	Meaning	Accepts Expression?
`stroke`	color, url, symbol	The color, graphic, or well-known shape to use to stroke lines or outlines	yes
`stroke-composite`	string	The composite mode to be used and the optional opacity separated with a comma. See the *full list of available modes*.	no
`stroke-geometry`	expression	An expression to use for the geometry when rendering features.	yes
`stroke-offset`	expression	Draws a parallel line using the specified distance, positive values offset left, negative right.	yes
`stroke-mime`	string (MIME Type)	The type of the image referenced by a url()	No, defaults to 'image/jpeg'
`stroke-opacity`	percentage	A value in the range of 0 (fully transparent) to 1.0 (fully opaque)	yes
`stroke-width`	length	The width to use for stroking the line.	yes
`stroke-size`	length	An image or symbol used for the stroke pattern will be stretched or squashed to this size before rendering. If this value differs from the stroke-width, the graphic will be repeated or clipped as needed.	yes
`stroke-rotation`	angle	A rotation to be applied (clockwise) to the stroke image. See also the stroke- repeat property.	yes
`stroke-linecap`	keyword: butt, square, round	The style to apply to the ends of lines drawn	yes
`stroke-linejoin`	keyword: miter, round, bevel	The style to apply to the "elbows" where segments of multiline features meet.	yes
`stroke-dasharray`	list of lengths	The lengths of segments to use in a dashed line.	no
`stroke-dashoffset`	length	How far to offset the dash pattern from the ends of the lines.	yes
`stroke-repeat`	keyword: repeat, stipple	How to use the provided graphic to paint the line. If repeat, then the graphic is repeatedly painted along the length of the line (rotated appropriately to match the line's direction). If stipple, then the line is treated as a polygon to be filled.	yes
`z-index`	integer	Controls the z ordering of output	no
`stroke-label-obstacle`	boolean	If true the line will be consider an obstable for labels, no label will overlap it	no

Polygon symbology

Property	Type	Meaning	Accepts Expression?
`fill`	color, url, symbol	The color, graphic, or well-known shape to use to stroke lines or outlines	yes
`fill-composite`	string	The composite mode to be used and the optional opacity separated with a comma. See the *full list of available modes*.	no
`fill-geometry`	expression	An expression to use for the geometry when rendering features.	yes
`fill-mime`	string (MIME Type)	The type of the image referenced by a url()	No, defaults to 'image/jpeg'
`fill-opacity`	percentage	A value in the range of 0 (fully transparent) to 1.0 (fully opaque)	yes
`fill-size`	length	The width to assume for the image or graphic provided.	yes
`fill-rotation`	angle	A rotation to be applied (clockwise) to the fill image.	yes
`z-index`	integer	Controls the z ordering of output	no
`fill-label-obstacle`	boolean	If true the polygon will be consider an obstable for labels, no label will overlap it	no
`graphic-margin`	List of lengths	A list of 1 to 4 values, specifying the space between repeated graphics in a texture paint. One value is uniform spacing in all directions, two values are considered top/bottom and right/left, three values are considered top, right/left, bottom, four values are read as top,right,bottom,left.	no
`random`	none,grid,free	Activates random distribution of symbols in a texture fill tile. See *Fills with randomized symbols* for details. Defaults to "none"	no
`random-seed`	integer number	The seed for the random generator. Defaults to 0	no
`random-rotation`	none/free	When set to "free" activates random rotation of the symbol in addition to random distribution. Defaults to "none"	no
`random-symbol-count`	positive integer number	Number of suymbols to be placed in the texture fill tile. May not be respected due to location conflicts (no two symbols are allowed to overlap). Defaults to 16.	no
`random-tile-size`	positive integer number	Size of the texture paint tile that will be filled with the random symbols. Defaults to 256.	no

Text symbology (labeling) - part 1

Property	Type	Meaning	Accepts Expression?
`label`	string	The text to display as labels for features	yes
`label-geometry`	expression	An expression to use for the geometry when rendering features.	yes
`label-anchor`	expression	The part of the label to place over the point or middle of the polygon. This takes 2 values - x y where x=0 is the left edge of the label, x=1 is the right edge. y=0 is the bottom edge of the label, y=1 is the top edge. Specify 0.5 0.5 to centre a label.	yes
`label-offset`	expression	This is for fine-tuning label-anchor. x and y values specify pixels to adjust the label position. For lines, a single value will make the label be parallel to the line, at the given distance, while two values will force a point style placement, with the label painted horizonally at the center of the line (plus the given offsets)	yes
`label-rotation`	expression	Clockwise rotation of label in degrees.	yes
`label-z-index`	expression	Used to determine which labels are drawn on top of other labels. Lower z-indexes are drawn on top.	yes
`shield`	mark, symbol	A graphic to display behind the label, such as a highway shield.	yes
`shield-mime`	string (MIME Type)	The type of the image referenced by a url()	No, defaults to 'image/jpeg'
`font-family`	string	The name of the font or font family to use for labels	yes
`font-fill`	fill	The fill to use when rendering fonts	yes
`font-style`	keyword: normal, italic, oblique	The style for the lettering	yes
`font-weight`	keyword: normal, bold	The weight for the lettering	yes
`font-size`	length	The size for the font to display.	yes
`halo-radius`	length	The size of a halo to display around the lettering (to enhance readability). This is *required* to activate the halo feature.	yes
`halo-color`	color	The color for the halo	yes
`halo-opacity`	percentage	The opacity of the halo, from 0 (fully transparent) to 1.0 (fully opaque).	yes
`label-padding`	length	The amount of 'padding' space to provide around labels. Labels will not be rendered closer together than this threshold. This is equivalent to the *spaceAround* vendor parameter.	no
`label-group`	one of: `true` or `false`	If true, the render will treat features with the same label text as a single feature for the purpose of labeling. This is equivalent to the *group* vendor parameter.	no
`label-max-displacement`	length	If set, this is the maximum displacement that the renderer will apply to a label. Labels that need larger displacements to avoid collisions will simply be omitted. This is equivalent to the *maxDisplacement* vendor parameter.	no

Text symbology (labeling) - part 2

Property	Type	Meaning	Accepts Expression?
`label-min-group-distance`	length	This is equivalent to the minGroupDistance vendor parameter in SLD.	no
`label-repeat`	length	If set, the renderer will repeat labels at this interval along a line. This is equivalent to the *repeat* vendor parameter.	no
`label-all-group`	one of `true` or `false`	when using grouping, whether to label only the longest line that could be built by merging the lines forming the group, or also the other ones. This is equivalent to the *allGroup* vendor parameter.	no
`label-remove-overlaps`	one of `true` or `false`	If enabled, the renderer will remove overlapping lines within a group to avoid duplicate labels. This is equivalent to the removeOverlaps vendor parameter.	no
`label-allow-overruns`	one of `true` or `false`	Determines whether the renderer will show labels that are longer than the lines being labelled. This is equivalent to the allowOverrun vendor parameter.	no
`label-follow-line`	one of `true` or `false`	If enabled, the render will curve labels to follow the lines being labelled. This is equivalent to the *followLine* vendor parameter.	no
`label-max-angle-delta`	one of `true` or `false`	The maximum amount of curve allowed between two characters of a label; only applies when 'follow-line: true' is set. This is equivalent to the *maxAngleDelta* vendor parameter.	no
`label-auto-wrap`	length	Labels will be wrapped to multiple lines if they exceed this length in pixels. This is equivalent to the *autoWrap* vendor parameter.	no
`label-force-ltr`	one of `true` or `false`	By default, the renderer will flip labels whose normal orientation would cause them to be upside-down. Set this parameter to false if you are using some icon character label like an arrow to show a line's direction. This is equivalent to the *forceLeftToRight* vendor parameter.	no
`label-conflict-resolution`	one of `true` or `false`	Set this to false to disable label conflict resolution, allowing overlapping labels to be rendered. This is equivalent to the *conflictResolution* vendor parameter.	no
`label-fit-goodness`	scale	The renderer will omit labels that fall below this "match quality" score. The scoring rules differ for each geometry type. This is equivalent to the *goodnessOfFit* vendor parameter.	no
`label-priority`	expression	Specifies an expression to use in determining which features to prefer if there are labeling conflicts. This is equivalent to the *Priority* SLD extension.	yes

Text symbology (labeling) - part 3

Property	Type	Meaning	Accepts Expression?
`shield-resize`	string, one of `none`, `stretch`, or `proportional`	Specifies a mode for resizing label graphics (such as highway shields) to fit the text of the label. The default mode, 'none', never modifies the label graphic. In `stretch` mode, GeoServer will resize the graphic to exactly surround the label text, possibly modifying the image's aspect ratio. In `proportional` mode, GeoServer will expand the image to be large enough to surround the text while preserving its original aspect ratio.	none
`shield-margin`	list of lengths, one to four elements long.	Specifies an extra margin (in pixels) to be applied to the label text when calculating label dimensions for use with the `shield-resize` option. Similar to the `margin` shorthand property in CSS for HTML, its interpretation varies depending on how many margin values are provided: 1 = use that margin length on all sides of the label 2 = use the first for top & bottom margins and the second for left & right margins. 3 = use the first for the top margin, second for left & right margins, third for the bottom margin. 4 = use the first for the top margin, second for the right margin, third for the bottom margin, and fourth for the left margin.	none
`label-underline`	one of `true` or `false`	If enabled, the renderer will underline labels. This is equivalent to the *underlineText* vendor parameter.	no
`label-strikethrough`	one of `true` or `false`	If enabled, the renderer will strikethrough labels. This is equivalent to the *strikethroughText* vendor parameter.	no
`label-char-spacing`	an amount of pixels, can be negative	If present, expands or shrinks the space between subsequent characters in a label according to the value specified	no
`label-word-spacing`	an amount of pixels, must be zero or positive	If present, expands the space between subsequent words in a label according to the value specified	no

Raster symbology

Property	Type	Meaning	Accepts Expression?
`raster-channels`	string	The list of raster channels to be used in the output. It can be "auto" to make the renderer choose the best course of action, or a list of band numbers, a single one will generate a gray image, three will generate an RGB one, four will generate a RGBA one. E.g., "1 3 7" to choose the first, third and seventh band of the input raster to make a RGB image	no
`raster-composite`	string	The composite mode to be used and the optional opacity separated with a comma. See the *full list of available modes*.	no
`raster-geometry`	expression	The attribute containing the raster to be painted. Normally not needed, but it would work if you had a custom vector data source that contains a GridCoverage attribute, in order to select it	yes
`raster-opacity`	floating point	A value comprised between 0 and 1, 0 meaning completely transparent, 1 meaning completely opaque. This controls the whole raster trasparency.	no
`raster-contrast-enhancement`	string	Allows to stretch the range of data/colors in order to enhance tiny differences. Possible values are 'normalize', 'histogram' and 'none'	no
`raster-gamma`	floating point	Gamma adjustment for the output raster	no
`raster-z-index`	integer	Controls the z ordering of the raster output	no
`raster-color-map`	string	Applies a color map to single banded input. The contents is a space separate list of `color-map-entry(color, value)` (opacity assumed to be 1), or `color-map-entry(color, value, opacity)`. The values must be provided in increasing order.	no
`raster-color-map-type`	string	Controls how the color map entries are interpreted, the possible values are "ramp", "intervals" and "values", with ramp being the default if no "raster-color-map-type" is provided. The default "ramp" behavior is to linearly interpolate color between the provided values, and assign the lowest color to all values below the lowest value, and the highest color to all values above the highest value. The "intervals" behavior instead assigns solid colors between values, whilst "values" only assigns colors to the specified values, every other value in the raster is not painted at all	no

Shared

Property	Type	Meaning	Accepts Expression?
`composite`	string	The composite mode to be used and the optional opacity separated with a comma. See the *full list of available modes*.	no
`composite-base`	one of `true` or `false`	This will tell the rendering engine to use that FeatureTypeStyle as the destination, and will compose all subsequent FeatureTypeStyle/Layers on top of it, until another base is found.	no
`geometry`	expression	An expression to use for the geometry when rendering features. This provides a geometry for all types of symbology, but can be overridden by the symbol-specific geometry properties.	yes
`sort-by`	string	A comma separated list of sorting directives, "att1 A|D, att2 A|D, ..." where `att?` are attribute names, and `A` or `D` are an optional direction specification, `A` is ascending, `D` is descending. Determines the loading, and thus painting, order of the features	no
`sort-by-group`	string	Rules with the different z-index but same sort-by-group id have their features sorted as a single group. Useful to z-order across layers or across different feature groups, like roads and rails, especially when using z-index to support casing	no
`transform`	function	Applies a rendering transformationon the current level. The function syntax is `txName(key1:value1,key1:value2)`. Values can be single ones, or space separated lists.	no

Symbol properties

These properties are applied only when styling built-in symbols. See *Styled marks* for details.

Property	Type	Meaning	Accepts Expression?
`size`	length	The size at which to render the symbol.	yes
`rotation`	angle	An angle through which to rotate the symbol.	yes

6.4.7 CSS value types

This page presents a brief overview of CSS types as used by this project. Note that these can be repeated as described in *Multi-valued properties*.

Numbers

Numeric values consist of a number, or a number annotated with a measurement value. In general, it is wise to use measurement annotations most of the time, to avoid ambiguity and protect against potential future changes to the default units.

Currently, the supported units include:

- Length
  - `px` pixels
  - `m` meters
  - `ft` feet
- Angle
  - `deg` degrees
- Ratio
  - `%` percentage

When using expressions in place of numeric values, the first unit listed for the type of measure is assumed.

Since the CSS module translates styles to SLD before any rendering occurs, its model of unit-of-measure is tied to that of SLD. In practice, this means that for any particular symbolizer, there only one unit-of-measure applied for the style. Therefore, the CSS module extracts that unit-of-measure from one special property for each symbolizer type. Those types are listed below for reference:

- `fill-size` determines the unit-of-measure for polygon symbolizers (but that doesn't matter so much since it is the only measure associated with fills)
- `stroke-width` determines the unit-of-measure for line symbolizers
- `mark-size` determines the unit-of-measure for point symbolizers
- `font-size` determines the unit-of-measure for text symbolizers and the associated halos

Strings

String values consist of a small snippet of text. For example, a string could be a literal label to use for a subset of roads:

```
[lanes>20] {
        label: "Serious Freaking Highway";
}
```

Strings can be enclosed in either single or double quotes. It's easiest to simply use whichever type of quotes are not in your string value, but you can escape quote characters by prefixing them with a backslash `\`. Backslash characters themselves must also be prefixed. For example, `'\\\''` is a string value consisting of a single backslash followed by a single single quote character.

Labels

While labels aren't really a special type of value, they deserve a special mention since labels are more likely to require special string manipulation than other CSS values.

If a label is a simple string value, then it works like any other string would:

```
[lanes > 20] {
    label: "Serious Freaking Highway";
}
```

However, if a label has multiple values, all of those values will be concatenated to form a single label:

```
[lanes > 20] {
   label: "Serious " "Freaking " "Highway";
}
```

Note the whitespace within the label strings here; *no whitespace is added* when concatenating strings, so you must be explicit about where you want it included. You can also mix CQL expressions in with literal string values here:

```
states {
   label: [STATE_NAME] " (" [STATE_ABBR] ")";
}
```

Note: This automatic concatenation is currently a special feature only provided for labels. However, string concatenation is also supported directly in CQL expressions by using the `strConcat` filter function:

```
* { fill: [strConcat('#', color_hex)]; }
```

This form of concatenation works with any property that supports expressions.

Colors

Color values are relatively important to styling, so there are multiple ways to specify them.

Format	Interpretation
`#RRGGBB`	A hexadecimal-encoded color value, with two digits each for red, green, and blue.
`#RGB`	A hexadecimal-encoded color value, with one digits each for red, green, and blue. This is equivalent to the two-digit-per-channel encoding with each digit duplicated.
`rgb(r, g, b)`	A three-part color value with each channel represented by a value in the range 0 to 1, or in the range 0 to 255. 0 to 1 is used if any of the values include a decimal point, otherwise it is 0 to 255.
Simple name	The simple English name of the color. A full list of the supported colors is available at http://www.w3.org/TR/SVG/types.html#ColorKeywords

External references

When using external images to decorate map features, it is necessary to reference them by URL. This is done by a call to the `url` function. The URL value may be wrapped in single or double-quotes, or not at all. The same escaping rules as for string values. The `url` function is

also a special case where the surrounding quote marks can usually be omitted. Some examples:

```
/* These properties are all equivalent. */

* {
    stroke: url("http://example.com/");
    stroke: url('http://example.com/');
    stroke: url(http://example.com/);
}
```

Note: While relative URLs are supported, they will be fully resolved during the conversion process to SLD and written out as absolute URLs. This may be cause problems when relocating data directories, etc. The style can be regenerated with the current correct URL by opening it in the demo editor and using the Submit button there.

Well-known marks

As defined in the SLD standard, GeoServer's `css` module also allows using a certain set of well-known mark types without having to provide graphic resources explicitly. These include:

- `circle`
- `square`
- `cross`
- `star`
- `arrow`

And others. Additionally, vendors can provide an extended set of well-known marks, a facet of the standard that is exploited by some GeoTools plugins to provide dynamic map features such as using characters from TrueType fonts as map symbols, or dynamic charting. In support of these extended mark names, the css module provides a `symbol` function similar to `url`. The syntax is the same, aside from the function name:

```
* {
    mark: symbol(circle);
    mark: symbol('ttf://Times+New+Roman&char=0x19b2');
    mark: symbol("chart://type=pie&x&y&z");
}
```

6.4.8 Directives

A directive is a CSS top level declaration that allows control of some aspects of the stylesheet application or translation to SLD. All directives are declared at the beginning of the CSS sheet and follow the same syntax:

```
@name value;
```

For example:

```
@mode 'Flat';
@styleTitle 'The title;
@styleAbstract 'This is a longer description'

* {
  stroke: black
}

[cat = 10] {
  stroke: yellow; stroke-width: 10
}
```

Supported directives

Directive	Type	Meaning	Accepts Expression?
`mode`	String, `Exclusive`, `Simple`, `Auto`, `Flat`	Controls how the CSS is translated to SLD. `Exclusive`, `Simple` and `Auto` are cascaded modes, `Flat` turns off cascading and has the CSS behave like a simplified syntax SLD sheet. See *Understanding Cascading in CSS* for an explanation of how the various modes work	false
`styleTitle`	String	The generated SLD style title	No
`styleAbstract`	String	The generated SLD style abstract/description	No

6.4.9 Understanding Cascading in CSS

Cascading Style Sheets are the styling language of the web, use a simple syntax, but sometimes their simplicity can be deceitful if the writer is not aware of how the "Cascading" part of it works. The confusion might become greater by looking at the translated SLD, and wondering how all the SLD rules came to be from a much smaller set of CSS rules.

This document tries to clarify how cascading works, how it can be controlled in SLD translation, and for those that would prefer simpler, if more verbose, styles, shows how to turn cascading off for good.

CSS rules application

Given a certain feature, how are CSS rules applied to it? This is roughly the algorithm:

- Locate all rules whose selector matches the current feature
- Sort them by specificity, less specific to more specific

- Have more specific rules add to and override properties set in less specific rules

  As you can see, depending on the feature attributes a new rule is built by the above algorithm, mixing all the applicable rules for that feature.

  The core of the algorithm allows to prepare rather succinct style sheets for otherwise very complex rule sets, by setting the common bits in less specific rules, and override them specifying the exceptions to the norm in more specific rules.

Understanding specificity

In web pages CSS specificity is setup as a tuple of four numbers called a,b,c,d:

- `a`: set to 1 if the style is local to an element, that is, defined in the element `style` attribute
- `b`: counts the number of ID attributes in the selector
- `c`: count the number of other attributes and pseudo classes in the selector
- `d`: count the number of element names or pseudo elements in the selector

`a` is more important than `b`, which is more important than `c`, and so on, so for example, if one rule has `a=1` and then second has `a=0`, the first is more specific, regardless of what values have `b`, `c` and `d`.

Here are some examples from the CSS specification, from less specific to more specific:

```
*
 ↪    {}  /* a=0 b=0 c=0 d=0 -> specificity = 0,0,0,0 */
li
 ↪    {}  /* a=0 b=0 c=0 d=1 -> specificity = 0,0,0,1 */
li:first-line␣
 ↪{}  /* a=0 b=0 c=0 d=2 -> specificity = 0,0,0,2 */
ul li
 ↪    {}  /* a=0 b=0 c=0 d=2 -> specificity = 0,0,0,2 */
ul ol+li
 ↪    {}  /* a=0 b=0 c=0 d=3 -> specificity = 0,0,0,3 */
h1 + *[rel=up]
 ↪{}  /* a=0 b=0 c=1 d=1 -> specificity = 0,0,1,1 */
ul ol li.red␣
 ↪ {}  /* a=0 b=0 c=1 d=3 -> specificity = 0,0,1,3 */
li.red.level␣
 ↪ {}  /* a=0 b=0 c=2 d=1 -> specificity = 0,0,2,1 */
#x34y
 ↪    {}  /* a=0 b=1 c=0 d=0 -> specificity = 0,1,0,0 */
style="..."␣
 ↪      /* a=1 b=0 c=0 d=0 -> specificity = 1,0,0,0 */
```

In cartographic CSS there are no HTML elements that could have a local style, so `a` is always zero. The others are calculated as follows:

- `b`: number of feature ids in the rule
- `c`: number of attributes in CQL filters and pseudo-classes (e.g., `:mark`) used in the selector
- `d`: 1 if a typename is specified, 0 otherwise

Here are some examples, from less to more specific:

```
*
↪    {}  /* a=0 b=0 c=0 d=0 -> specificity = 0,0,0,0 */
topp:states
↪    {}  /* a=0 b=0 c=0 d=1 -> specificity = 0,0,0,1 */
:mark
↪    {}  /* a=0 b=0 c=1 d=0 -> specificity = 0,0,1,0 */
[a = 1 and b > 10]
↪{}  /* a=0 b=0 c=1 d=0 -> specificity = 0,0,2,0 */
#states.1
↪    {}  /* a=0 b=1 c=0 d=0 -> specificity = 0,1,0,0 */
```

In case two rules have the same specificity, the last one in the document wins.

Understanding CSS to SLD translation in cascading mode

As discussed above, CSS rule application can potentially generate a different rule for each feature, depending on its attributes and how they get matched by the various CSS selectors.

SLD on the other hand starts from the rules, and applies all of them, in turn, to each feature, painting each matching rule. The two evaluation modes are quite different, in order to turn CSS into SLD the translator has to generate every possible CSS rule combination, while making sure the generated SLD rules are mutually exclusive (CSS generated a single rule for a given feature in the end).

The combination of all rules is called a power set, and the exclusivity is guaranteed by negating the filters of all previously generated SLD rules and adding to the current one. As one might imagine, this would result in a lot of rules, with very complex filters.

The translator addresses the above concerns by applying a few basic strategies:

- The generated filters are evaluated in memory, if the filter is found to be "impossible", that is, something that could never match an exiting feature, the associated rule is not emitted (e.g., `a = 1 and a = 2` or `a = 1 and not(a = 1)`)
- The generated SLD has a vendor option `<sld:VendorOption name="ruleEvaluation">first</sld:VendorOption>` which forces the renderer to give up evaluating further rules once one of them actually matched a feature

The above is nice and sufficient in theory, while in practice it can break down with very complex CSS styles having a number of orthogonal selectors (e.g., 10 rules controlling the fill on the values of attribute `a` and 10 rules controlling the stroke on values of attribute `b`, and another 10 rules controlling the opacity of fill and stroke based on attribute `c`, resulting in 1000 possible combinations).

For this reason by default the translator will try to generated simplified and fully exclusive rules only if the set of rules is "small", and will instead generate the full power set otherwise, to avoid incurring in a CSS to SLD translation time of minutes if not hours.

The translation modes are controlled by the `@mode` directive, with the following values:

- `'Exclusive'`: translate the style sheet in a minimum set of SLD rules with simplified selectors, taking whatever time and memory required
- `'Simple'`: just generated the power set without trying to build a minimum style sheet, ensuring the translation is fast, even if the resulting SLD might look very complex
- `'Auto'`: this is the default value, it will perform the power set expansion, and then will proceed in `Exclusive` mode if the power set contains less than 100 derived rules, or in `Simple` mode otherwise. The rule count threshold can be manually controlled by using the `@autoThreshold` directive.

The Flat translation mode

The `@mode` directive has one last possible value, `Flat`, which enables a flat translation mode in which specificity and cascading are not applied.

In this mode the CSS will be translated almost 1:1 into a corresponding SLD, each CSS rule producing and equivalent SLD rule, with the exception of the rules with pseudo-classes specifying how to stroke/fill marks and symbols in general.

Care should be taken when writing rules with pseudo classes, they will be taken into consideration only if their selector matches the one of the preceding rule. Consider this example:

```
@mode "Flat";

[type = 'Capital'] {
  mark: symbol(circle);
}

[type = 'Capital'] :mark {
  fill: white;
  size: 6px;
}

:mark {
  stroke: black;
  stroke-width: 2px;
}
```

In the above example, the first rule with the `:mark` pseudo class will be taken into consideration and merged with the capital one, the second one instead will be ignored. The resulting SLD will thus not contain any stroke specification for the 'circle' mark:

```
<?xml version=
↪"1.0" encoding="UTF-8"?><sld:StyledLayerDescriptor␣
↪xmlns="http://www.opengis.net/sld"
      xmlns:sld="http://www.opengis.
↪net/sld" xmlns:ogc="http://www.opengis.net/ogc"
      xmlns:gml=
↪"http://www.opengis.net/gml" version="1.0.0">
  <sld:NamedLayer>
    <sld:Name/>
```

```
    <sld:UserStyle>
      <sld:Name>Default Styler</sld:Name>
      <sld:FeatureTypeStyle>
        <sld:Rule>
          <ogc:Filter>
            <ogc:PropertyIsEqualTo>
              <ogc:PropertyName>type</ogc:PropertyName>
              <ogc:Literal>Capital</ogc:Literal>
            </ogc:PropertyIsEqualTo>
          </ogc:Filter>
          <sld:PointSymbolizer>
            <sld:Graphic>
              <sld:Mark>
                <sld:WellKnownName>circle</sld:WellKnownName>
                  <sld:Fill>
                    <sld:CssParameter name="fill">#ffffff</sld:CssParameter>
                  </sld:Fill>
                </sld:Mark>
                <sld:Size>6</sld:Size>
              </sld:Graphic>
            </sld:PointSymbolizer>
          </sld:Rule>
        </sld:FeatureTypeStyle>
      </sld:UserStyle>
    </sld:NamedLayer>
  </sld:StyledLayerDescriptor>
```

The advantages of flat mode are:

- Easy to understand, the rules are applied in the order they are written
- Legend control, the generated legend contains no surprises as rules are not mixed together and are not reordered

  The main disadvantage is that there is no more a way to share common styling bits in general rules, all common bits have to be repeated in all rules.

Note: In the future we hope to add the ability to nest rules, which is going to address some of the limitations of flat mode without introducing the most complex bits of the standard cascading mode

Comparing cascading vs flat modes, an example

Consider the following CSS:

```
* { stroke: black; stroke-width: 10 }

[cat = 'important'] { stroke: yellow; }
```

If the above style is translated in cascading mode, it will generate two mutually exclusive SLD rules:

- One applying a 10px wide yellow stroke on all features whose cat attribute is 'important'
- One applying a 10px wide black stroke on all feature whose cat attribute is not 'important'

  Thus, each feature will be painted by a single line, either yellow or black.

  If instead the style contains a `@mode 'Flat'` directive at the top, it will generated two non mutually exclusive SLD rules:

- One applying a 10px wide black stroke on all features
- One applying a 1px wide yewllow stroke on all feature whose cat attribute is 'important'

  Thus, all features will at least be painted 10px black, but the 'important' ones will also have a second 1px yellow line *on top of the first one*

6.4.10 Nested rules

Starting with GeoServer 2.10 the CSS modules supports rule nesting, that is, a child rule can be written among properties of a parent rule. The nested rules inherits the parent rule selector and properties, adding its own extra selectors and property overrides.

Each nested rule can be written as normal, however, if other rules or properties follow, it must be terminated with a semicolon (this char being the separator in the CSS language).

Nesting is a pure syntax improvement, as such it does not actually provide extra functionality, besides more compact and hopefully readable styles.

This is an example of a CSS style using only cascading to get a different shape, fill and stroke color for a point symbol in case the `type` attribute equals to `important`:

```
[@sd < 3000] {
  mark: symbol(circle)
}

[@sd < 3000] :mark {
  fill: gray;
  size: 5
}

[@sd < 3000] [type = 'important'] {
  mark: symbol('triangle')
}

[@sd < 3000] [type = 'important'] :mark {
  fill: red;
  stroke: yellow
}
```

This second version uses rule nesting getting a more compact expression, putting related symbology element close by:

```
[@sd < 3000] {
   mark: symbol(circle);
   :mark {
      fill: gray;
      size: 5
   };
   [type = 'important'] {
      mark: symbol(triangle);
      :mark {
        fill: red;
        stroke: yellow
      }
   }
}
```

6.4.11 Rendering transformations in CSS

Starting with GeoServer 2.10 the CSS modules supports rendering transformations via the `transform` property.

The property is a function call with a special key/value pair syntax, using the following template:

```
transformationName(key1:value1,
↪key2:v21 v22 ... v2M,...,keyN:vN)
```

The values can be simple ones, or can be a space separated list. The parameter representing the input layer can be omitted, the engine will automatically recognize input parameters of type feature collection or grid coverage.

The transformation function is subject to cascading like all other properties, but cascading acts at the whole z-level, so if multiple transformations are needed, they need to be associated with two different z-levels.

This is an example of a CSS style extracting contour lines from a DEM, and also showing the single values when a suitable zoom level is reached:

```
/* @title Levels */
* {
  transform: ras:Contour(levels:␣
↪1100 1200 1300 1400 1500 1600 1700);
  z-index: 0;
  stroke: gray;
  label: [numberFormat('#', value)];
  font-size: 12;
  font-fill: black;
  font-weight: bold;
  halo-color: white;
  halo-radius: 2;
  label-follow-line: true;
  label-repeat: 200;
```

```
  label-max-angle-delta: 45;
  label-priority: 2000;
}

/* @title Values */
[@sd < 12000] {
  transform: ras:RasterAsPointCollection(scale: 0.5);
  z-index: 1;
  label: [GRAY_INDEX];
  label-anchor: 0.5 0.5;
  font-family: Arial;
  font-fill: black;
  font-size: 6;
  label-priority: 1000;
}
```

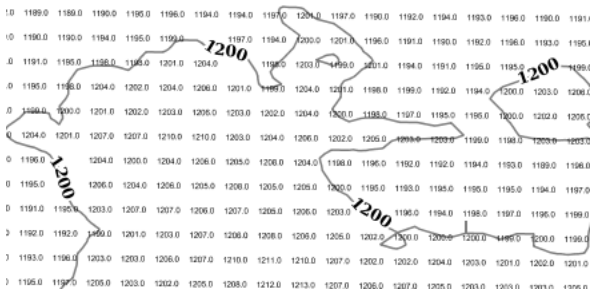


Fig. 6.110: *The two transformations in action against a DEM layer*

6.4.12 Styled marks

GeoServer's CSS module provides a collection of predefined symbols that you can use and combine to create simple marks, strokes, and fill patterns without needing an image editing program. You can access these symbols via the symbol() CSS function. For example, the built-in circle symbol makes it easy to create a simple 'dot' marker for a point layer:

```
* {
  mark: symbol(circle);
}
```

Symbols work anywhere you can use a `url()` to reference an image (as in, you can use symbols for stroke and fill patterns as well as markers.)

Symbol names

GeoServer extensions can add extra symbols (such as the `chart://` symbol family which allows the use of charts as symbols via a naming scheme similar to the Google Charts API). However, there are a few symbols that are always available:

- circle

- square
- triangle
- arrow
- cross
- star
- x
- shape://horizline
- shape://vertline
- shape://backslash
- shape://slash
- shape://plus
- shape://times
- windbarbs://default(size)[unit]

Symbol selectors

Symbols offer some additional styling options beyond those offered for image references. To specify these style properties, just add another rule with a special selector. There are 8 "pseudoclass" selectors that are used to style selectors:

- `:mark` specifies that a rule applies to symbols used as point markers
- `:shield` specifies that a rule applies to symbols used as label shields (icons displayed behind label text)
- `:stroke` specifies that a rule applies to symbols used as stroke patterns
- `:fill` specifies that a rule applies to symbols used as fill patterns
- `:symbol` specifies that a rule applies to any symbol, regardless of which context it is used in
- `:nth-mark(n)` specifies that a rule applies to the symbol used for the nth stacked point marker on a feature.
- `:nth-shield(n)` specifies that a rule applies to the symbol used for the background of the nth stacked label on a feature
- `:nth-stroke(n)` specifies that a rule applies to the symbol used for the nth stacked stroke pattern on a feature.
- `:nth-fill(n)` specifies that a rule applies to the symbol used for the nth stacked fill pattern on a feature.
- `:nth-symbol(n)` specifies that a rule applies to the symbol used for the nth stacked symbol on a feature, regardless of which context it is used in.

These pseudoclass selectors can be used in a top level rule, but starting with GeoServer 2.10, they are more commonly used in sub-rules close to the mark property, to get better readability (see example below).

Symbol styling properties

Styling a built-in symbol is similar to styling a polygon feature. However, the styling options are slightly different from those available to a true polygon feature:

- The `mark` and `label` families of properties are unavailable for symbols.
- Nested symbol styling is not currently supported.
- Only the first `stroke` and `fill` will be used.
- Additional `size` (as a length) and `rotation` (as an angle) properties are available. These are analogous to the `(mark|stroke|fill)-size` and `(mark|stroke|fill)-rotation` properties available for true geometry styling.

Note: The various prefixed '-size' and '-rotation' properties on the containing style override those for the symbol if they are present.

Example styled symbol

As an example, consider a situation where you are styling a layer that includes data about hospitals in your town. You can create a simple hospital logo by placing a red cross symbol on top of a white circle background:

```
[usage='hospital'] {
  mark: symbol('circle'), symbol('cross');
  :nth-mark(1) {
    size: 16px;
    fill: white;
    stroke: red;
  };
  :nth-mark(2) {
    size: 12px;
    fill: red;
  }
}
```

Also an windbarb example where you get wind speed and direction from your data fields horSpeed and horDir (direction):

```
* {
  /* select windbard based on speed(
↪here in meters per second, and south hemisphere) */
  mark: symbol('windbarbs:/
↪/default(${horSpeed})[m/s]?hemisphere=s');

  /* rotate
↪windbarb based on horDir property (in degrees) */
  mark-rotation: [horDir];
```

```
  mark-size: 20;
}
```

6.4.13 CSS Cookbook

The CSS Cookbook is a collection of CSS "recipes" for creating various types of map styles. Wherever possible, each example is designed to show off a single CSS feature so that code can be copied from the examples and adapted when creating CSS styles of your own. Most examples are shared with the SLD Cookbook, to make a comparison between the two syntaxes immediate.

The CSS Cookbook is divided into four sections: the first three for each of the vector types (points, lines, and polygons) and the fourth section for rasters. Each example in every section contains a screenshot showing actual GeoServer WMS output and the full CSS code for reference.

Each section uses data created especially for the Cookbooks (both CSS and SLD), with shapefiles for vector data and GeoTIFFs for raster data. The projection for data is EPSG:4326. All files can be easily loaded into GeoServer in order to recreate the examples.

Data type	Shapefile
Point	`sld_cookbook_point.zip`
Line	`sld_cookbook_line.zip`
Polygon	`sld_cookbook_polygon.zip`
Raster	`sld_cookbook_raster.zip`

Points

While points are seemingly the simplest type of shape, possessing only position and no other dimensions, there are many different ways that a point can be styled in CSS.

Example points layer

The `points layer` used for the examples below contains name and population information for the major cities of a fictional country. For reference, the attribute table for the points in this layer is included below.

fid (Feature ID)	**name** (City name)	**pop** (Population)
point.1	Borfin	157860
point.2	Supox City	578231
point.3	Ruckis	98159
point.4	Thisland	34879
point.5	Synopolis	24567
point.6	San Glissando	76024
point.7	Detrania	205609

`Download the points shapefile`

Simple point

This example specifies points be styled as red circles with a diameter of 6 pixels.

Fig. 6.111: *Simple point*

Code

```
* {
  mark: symbol(circle);
  mark-size: 6px;
  :mark {
    fill: red;
  }
}
```

Details

There are two rules in this CSS, the outer one matches all features, and asks them to be depicted with a circular mark, 6 pixels wide. The nested rule uses a symbol selector, `:mark`, which selects all marks, and allows to specify how to fill the contents of the circle, in this case, with a solid red fill (a stand alone fill property would have been interpreted as the request to fill all polygons in the input with solid red instead).

Simple point with stroke

This example adds a stroke (or border) around the *Simple point*, with the stroke colored black and given a thickness of 2 pixels.

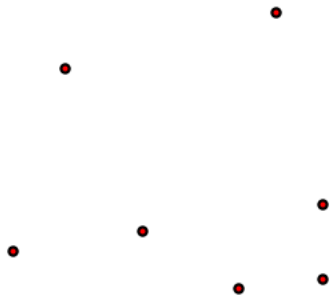


Fig. 6.112: *Simple point with stroke*

Code

```
* {
  mark: symbol(circle);
  mark-size: 6px;
  :mark {
    fill: red;
    stroke: black;
    stroke-width: 2px;
  }
}
```

Details

This example is similar to the *Simple point* example, in this case a stroke and a stroke width have been specified in the mark selector in order to apply them to the circle symbols.

Rotated square

This example creates a square instead of a circle, colors it green, sizes it to 12 pixels, and rotates it by 45 degrees.

Code

```
* {
  mark: symbol(square);
  mark-size: 12px;
  mark-rotation: 45;
  :mark {
    fill: #009900;
  }
}
```

Fig. 6.113: *Rotated square*

Details

In this example, **line 2** sets the shape to be a square, with **line 6** setting the color to a dark green (`#009900`). **Line 3** sets the size of the square to be 12 pixels, and **line 4** set the rotation is to 45 degrees.

Transparent triangle

This example draws a triangle, creates a black stroke identical to the *Simple point with stroke* example, and sets the fill of the triangle to 20% opacity (mostly transparent).

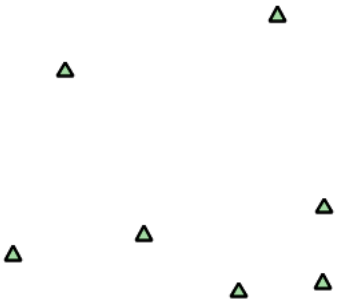


Fig. 6.114: *Transparent triangle*

Code

```
1  * {
2    mark: symbol(triangle);
3    mark-size: 12;
4    :mark {
5      fill: #009900;
```

```
6                 fill-opacity: 0.2;
7                 stroke: black;
8                 stroke-width : 2px;
9               }
10            }
```

Details

In this example, **line 2** once again sets the shape, in this case to a triangle, where **line 3** sets the mark size to 12 pixels. **Line 5** sets the fill color to a dark green (`#009900`) and **line 6** sets the opacity to 0.2 (20% opaque). An opacity value of 1 means that the shape is drawn 100% opaque, while an opacity value of 0 means that the shape is drawn 0% opaque, or completely transparent. The value of 0.2 (20% opaque) means that the fill of the points partially takes on the color and style of whatever is drawn beneath it. In this example, since the background is white, the dark green looks lighter. Were the points imposed on a dark background, the resulting color would be darker. **Line 8** set the stroke color to black and width to 2 pixels.

Point as graphic

This example styles each point as a graphic instead of as a simple shape.

Fig. 6.115: *Point as graphic*

Code

```
1             * {
2               mark: url(smileyface.png);
3               mark-mime: "image/png";
4             }
```

Details

This style uses a graphic instead of a simple shape to render the points. **Line 2** sets the path and file name of the graphic, while **line 3** indicates the format (MIME type) of the graphic (image/png). In this example, the graphic is contained in the same directory as the SLD, so no path information is necessary, although a full URL could be used if desired.

Fig. 6.116: *Graphic used for points*

Point with default label

This example shows a text label on the *Simple point* that displays the "name" attribute of the point. This is how a label will be displayed in the absence of any other customization.

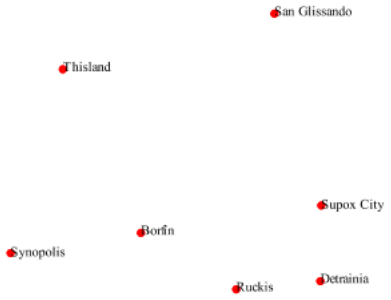


Fig. 6.117: *Point with default label*

Code

```
* {
  mark: symbol(circle);
  mark-size: 6px;
  label: [name];
  font-fill: black;
  :mark {
    fill: red;
  }
}
```

Details

This style is quite similar to the *Simple point*, but two new properties have been added to specify the labelling options. **Line 4** indicates that the label contents come from the "name" attribute (anything in square brackets is a CQL expression, the attribute name being the simplest case) while **Line 5** sets the label color to black.

Point with styled label

This example improves the label style from the *Point with default label* example by centering the label above the point and providing a different font name and size.

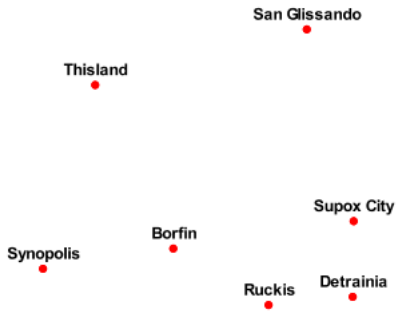


Fig. 6.118: *Point with styled label*

Code

```
1  * {
2    mark: symbol(circle);
3    mark-size: 6px;
4    label: [name];
5    font-fill: black;
6    font-family: Arial;
7    font-size: 12;
8    font-weight: bold;
9    label-anchor: 0.5 0;
10   label-offset: 0 5;
11   :mark {
12     fill: red;
13   }
14
15 }
```

Details

This example expands on *Point with default label* and specifies the font attributes, in particular, the text is Aria, bold, 12px wide. Moreover, the label is moved on top of the point, by specifying an anchor of `0.5 0`, which sets the point to be centered (0.5) horizontally axis and bottom aligned (0.0) vertically with the label, and an offset which moves the label 5 pixels up vertically.

The result is a centered bold label placed slightly above each point.

Point with rotated label

This example builds on the previous example, *Point with styled label*, by rotating the label by 45 degrees, positioning the labels farther away from the points, and changing the color of the label to purple.

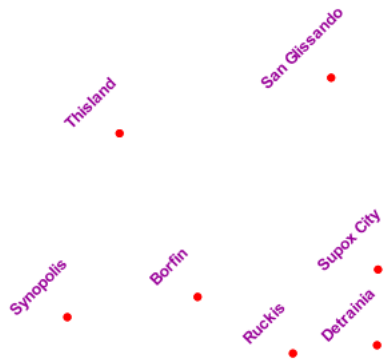


Fig. 6.119: *Point with rotated label*

Code

```
* {
  mark: symbol(circle);
  mark-size: 6px;
  label: [name];
  font-fill: #990099;
  font-family: Arial;
  font-size: 12;
  font-weight: bold;
  label-anchor: 0.5 0;
  label-offset: 0 25;
  label-rotation: -45;
  :mark {
    fill: red;
  }
}
```

Details

This example is similar to the *Point with styled label*, but there are three important differences. **Line 10** specifies 25 pixels of vertical displacement. **Line 11** specifies a rotation of "-45" or 45 degrees counter-clockwise. (Rotation values increase clockwise, which is why the value is negative.) Finally, **line 5** sets the font color to be a shade of purple (`#990099`).

Note that the displacement takes effect before the rotation during rendering, so in this example, the 25 pixel vertical displacement is itself rotated 45 degrees.

Attribute-based point

This example alters the size of the symbol based on the value of the population ("pop") attribute.

Fig. 6.120: *Attribute-based point*

Code

```
1  * {
2    mark: symbol(circle);
3    :mark {
4      fill: #0033CC;
5    };
6    [pop < 50000] {
7      mark-size: 8;
8    };
9    [pop >= 50000] [pop < 100000] {
10     mark-size: 12;
11   };
12   [pop >= 100000] {
13     mark-size: 16;
14   }
15 }
```

Details

Note: Refer to the *Example points layer* to see the attributes for this data. This example has eschewed labels in order to simplify the style, but you can refer to the example *Point with styled label* to see which attributes correspond to which points.

This style shows how the basic mark setup (red circle, default size) can be overridden via cascading/nesting, changing the size depending on the pop attribute value, with smaller values yielding a smaller circle, and larger values yielding a larger circle.

The three rules are designed as follows:

Rule order	**Rule name**	**Population** ("pop")	**Size**
1	SmallPop	Less than 50,000	8
2	MediumPop	50,000 to 100,000	12
3	LargePop	Greater than 100,000	16

The result of this style is that cities with larger populations have larger points. In particular, the rule at **Line 6** matches all features whose "pop" attribute is less than 50000, the rule at **Line 9** matches all features whose "pop" attribute is between 50000 and 100000 (mind the space between the two predicates, it is equivalent to and AND, if we had used a comma it would have been an OR instead), while the rule at **Line 12** matches all features with more than 100000 inhabitants.

Zoom-based point

This example alters the style of the points at different zoom levels.

Code

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
```

```
* {
  mark: symbol(circle);
}

:mark {
  fill: #CC3300;
}

[@sd < 16M] {
  mark-size: 12;
}

[@sd > 16M] [@sd < 32M] {
  mark-size: 8;
}

[@sd > 32M] {
```

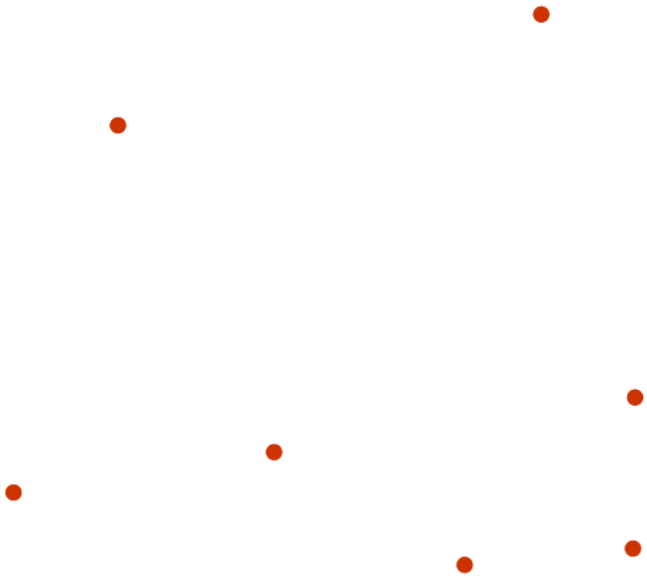


Fig. 6.121: *Zoom-based point: Zoomed in*

Fig. 6.122: *Zoom-based point: Partially zoomed*

Fig. 6.123: *Zoom-based point: Zoomed out*

```
18                                    mark-size: 4;
19                                }
```

Details

It is often desirable to make shapes larger at higher zoom levels when creating a natural-looking map. This example styles the points to vary in size based on the zoom level (or more accurately, scale denominator). Scale denominators refer to the scale of the map. A scale denominator of 10,000 means the map has a scale of 1:10,000 in the units of the map projection.

Note: Determining the appropriate scale denominators (zoom levels) to use is beyond the scope of this example.

This style contains three rules matching the scale. The three rules are designed as follows:

Rule order	Rule name	Scale denominator	Point size
1	Large	1:16,000,000 or less	12
2	Medium	1:16,000,000 to 1:32,000,000	8
3	Small	Greater than 1:32,000,000	4

The order of these rules does not matter since the scales denominated in each rule do not overlap.

The rules use the "@sd" pseudo-attribute, which refers to the current scale denominator, and which can be compared using the '<' and '>' operators only (using any other operator or function will result in errors).

The result of this style is that points are drawn larger as one zooms in and smaller as one zooms out.

While this example uses on purpose cascading to show a different possible setup, the same style could be written as:

```
* {
  mark: symbol(circle);
  :mark {
    fill: #CC3300;
  };
  [@sd < 16M] {
    mark-size: 12;
  };
  [@sd > 16M] [@sd < 32M] {
    mark-size: 8;
  };
  [@sd > 32M] {
    mark-size: 4;
  }
}
```

Lines

While lines can also seem to be simple shapes, having length but no width, there are many options and tricks for making lines display nicely.

Example lines layer

The `lines layer` used in the examples below contains road information for a fictional country. For reference, the attribute table for the points in this layer is included below.

fid (Feature ID)	**name** (Road name)	**type** (Road class)
line.1	Latway	highway
line.2	Crescent Avenue	secondary
line.3	Forest Avenue	secondary
line.4	Longway	highway
line.5	Saxer Avenue	secondary
line.6	Ridge Avenue	secondary
line.7	Holly Lane	local-road
line.8	Mulberry Street	local-road
line.9	Nathan Lane	local-road
line.10	Central Street	local-road
line.11	Lois Lane	local-road
line.12	Rocky Road	local-road
line.13	Fleet Street	local-road
line.14	Diane Court	local-road
line.15	Cedar Trail	local-road
line.16	Victory Road	local-road
line.17	Highland Road	local-road
line.18	Easy Street	local-road
line.19	Hill Street	local-road
line.20	Country Road	local-road
line.21	Main Street	local-road
line.22	Jani Lane	local-road
line.23	Shinbone Alley	local-road
line.24	State Street	local-road
line.25	River Road	local-road

Download the lines shapefile

Simple line

This example specifies lines be colored black with a thickness of 3 pixels.

Code

```
* {
  stroke: black;
  stroke-width: 3px;
}
```

Details

The only rule asks for a black stroke (this attribute is mandatory to get strokes to actually show up), 3 pixels wide.

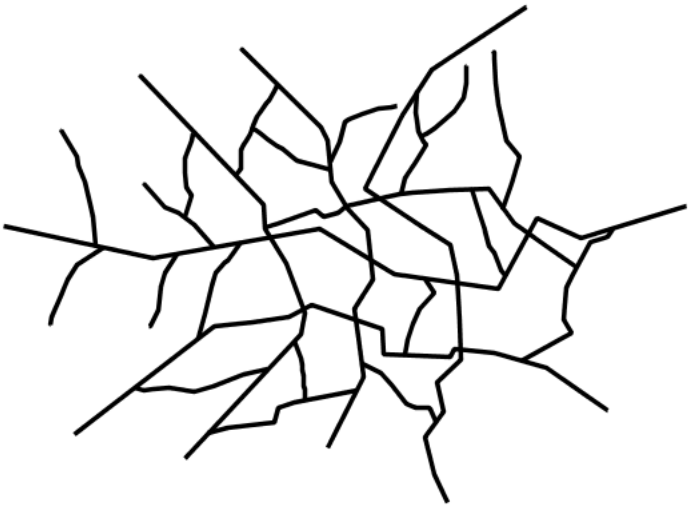


Fig. 6.124: *Simple line*

Line with border

This example shows how to draw lines with borders (sometimes called "cased lines"). In this case the lines are drawn with a 3 pixel blue center and a 1 pixel wide gray border.

Code

```
* {
  stroke: #333333, #6699FF;
  stroke-width: 5px, 3px;
  stroke-linecap: round;
  z-index: 0, 1;
}
```

Details

Lines in CSS have no notion of a "fill", only "stroke". Thus, unlike points or polygons, it is not possible to style the "edge" of the line geometry. It is, however, possible to achieve this effect by drawing each line twice: once with a certain width and again with a slightly smaller width. This gives the illusion of fill and stroke by obscuring the larger lines everywhere except along the edges of the smaller lines.

The style uses the "multi-valued properties" CSS support by specifying two strokes and two stroke-widths. This causes each feature to be painted twice, first with a dark gray (`#333333`) line 5 pixels wide, and then a thinner blue (`#6699FF`) line 3 pixels wide.

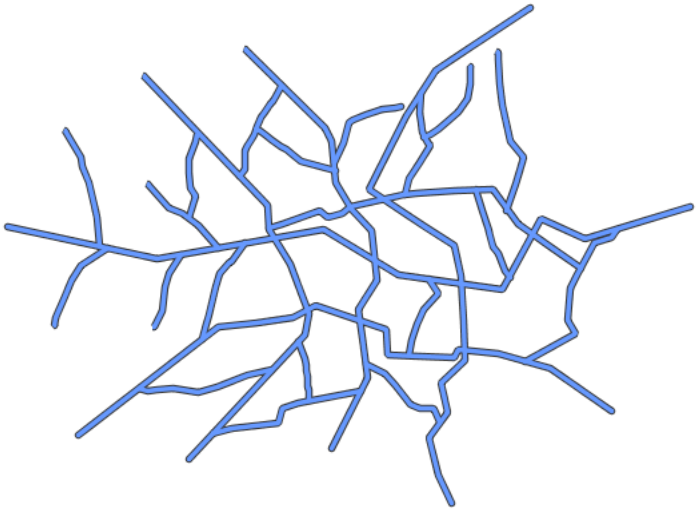


Fig. 6.125: *Line with border*

Since every line is drawn twice, the order of the rendering is *very* important. Without the z-index indication, each feature would first draw the gray stroke and then the blue one, and then the rendering engine would move to the next feature, and so on. This would result in ugly overlaps when lines do cross. By using the z-index property (**Line 3**) instead, all gray lines will be painted first, and then all blue lines will painted on top, thus making sure the blue lines visually connect.

The "stroke-linecap" property is the only one having a single value, this is because the value is the same for both the gray and blue line.

The result is a 3 pixel blue line with a 1 pixel gray border, since the 5 pixel gray line will display 1 pixel on each side of the 3 pixel blue line.

Dashed line

This example alters the *Simple line* to create a dashed line consisting of 5 pixels of drawn line alternating with 2 pixels of blank space.

Code

```
* {
  stroke: blue;
  stroke-width: 3px;
  stroke-dasharray: 5 2;
}
```

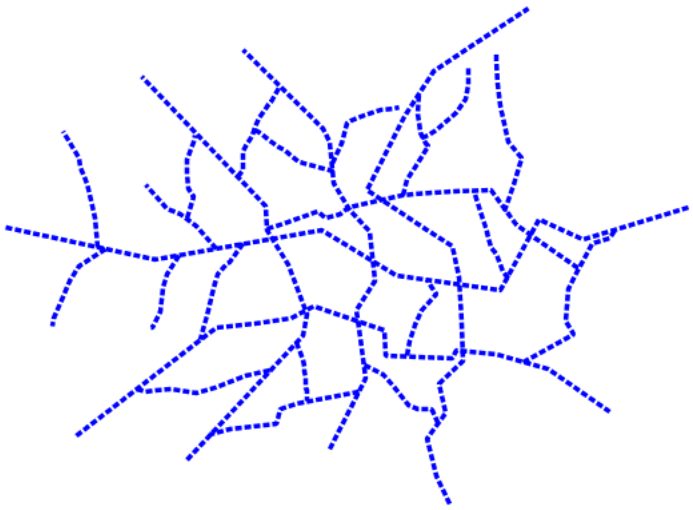


Fig. 6.126: *Dashed line*

Details

In this example the we create a blue line, 3 pixels wide, and specify a dash array with value "5 2", which creates a repeating pattern of 5 pixels of drawn line, followed by 2 pixels of omitted line.

Railroad (hatching)

This example uses hatching to create a railroad style. Both the line and the hatches are black, with a 2 pixel thickness for the main line and a 1 pixel width for the perpendicular hatches.

Code

```
* {
  stroke: #333333, symbol("shape://vertline");
  stroke-width: 3px;
  :nth-stroke(2) {
    size: 12;
    stroke: #333333;
    stroke-width: 1px;
  }
}
```

Details

In this example a multi-valued stroke is used: the fist value makes the renderer paint a dark gray line (3 pixels wide, according to the

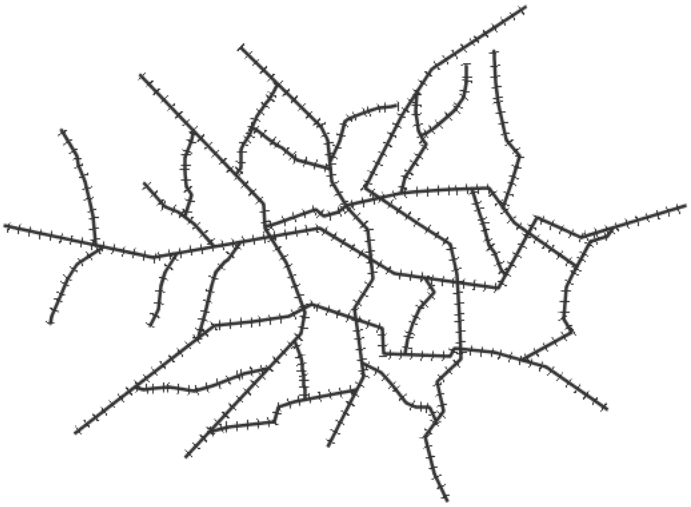


Fig. 6.127: *Railroad (hatching)*

"stroke-width" attribute), whilst the second value makes the line be painted by repeating the "shape://vertline" symbol over and over, creating the hatching effect.

In order to specify how the symbol itself should be painted, the ":nth-stroke(2)" pseudo-selector is used at **Line 4** to specify the options for the repeated symbol: in particular with are instructing the renderer to create a 12px wide symbol, with a dark gray stroke 1 pixel wide.

Spaced graphic symbols

This example uses a graphic stroke along with dash arrays to create a "dot and space" line type. Adding the dash array specification allows to control the amount of space between one symbol and the next one. Without using the dash array the lines would be densely populated with dots, each one touching the previous one.

Code

```
1  * {
2    stroke: symbol(circle);
3    stroke-dasharray: 4 6;
4    :stroke {
5      size: 4;
6      fill: #666666;
7      stroke: #333333;
8      stroke-width: 1px;
9    }
10 }
```

Fig. 6.128: *Spaced symbols along a line*

Details

This example, like others before, uses `symbol(circle)` to place a graphic symbol along a line.

The symbol details are specified in the nested rule at **Line 4** using the ":stroke" pseudo-selector, creating a gray fill circle, 4 pixels wide, with a dark gray outline.

The spacing between symbols is controlled with the `stroke-dasharray` at **line 3**, which specifies 4 pixels of pen-down (just enough to draw the circle) and 6 pixels of pen-up, to provide the spacing.

Alternating symbols with dash offsets

This example shows how to create a complex line style which alternates a dashed line and a graphic symbol. The code builds on features shown in the previous examples:

- `stroke-dasharray` controls pen-down/pen-up behavior to generate dashed lines
- `symbol(...)` places symbols along a line combining the two allows control of symbol spacing

This also shows the usage of a *dash offset*, which controls where rendering starts in the dash array. For example, with a dash array of `5 10` and a dash offset of `7` the renderer starts drawing the pattern 7 pixels from the beginning. It skips the 5 pixels pen-down section and 2 pixels of the pen-up section, then draws the remaining 8 pixels of pen-up, then 5 down, 10 up, and so on.

The example shows how to use these features to create two synchronized sequences of dash arrays, one drawing line segments and the other symbols.

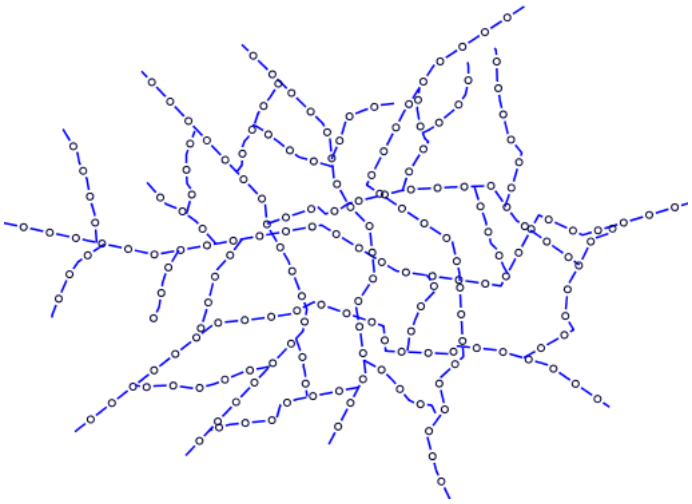


Fig. 6.129: *Alternating dash and symbol*

Code

```
1  * {
2    stroke: blue, symbol(circle);
3    stroke-width: 1px;
4    stroke-dasharray: 10 10, 5 15;
5    stroke-dashoffset: 0, 7.5;
6    :nth-stroke(2) {
7      stroke: #000033;
8      stroke-width: 1px;
9      size: 5px;
10   }
11 }
```

Details

This example uses again multi-valued properties
to create two subsequent strokes applied to the same lines.
The first
stroke is a solid blue line, 1 pixel wide, with a dash array of "10 10".
The second one instead is a repeated
circle, using a dash array of "5 15" and with a dash offset of
7.5. This makes the sequence start with 12.5 pixels of white space,
then a circle (which is then centered between the two line segments
of the other pattern), then 15 pixels of white space, and so on.

The circle portrayal details are specified using the pseudo selector "nth-stroke(2)" at **line 6**, asking for circles that are 5 pixels wide, not filled, and with a dark blue outline.

Line with default label

This example shows a text label on the simple line. This is how a label will be displayed in the absence of any other customization.

Fig. 6.130: *Line with default label*

Code

```
* {
  stroke: red;
  label: [name];
  font-fill: black;
}
```

Details

This example paints lines with a red stroke, and then adds horizontal black labels at the center of the line, using the "name" attribute to fill the label.

_css_line_

Labels along line with perpendicular offset

This example shows a text label on the simple line, just like the previous example, but will force the label to be parallel to the lines, and will offset them a few pixels away.

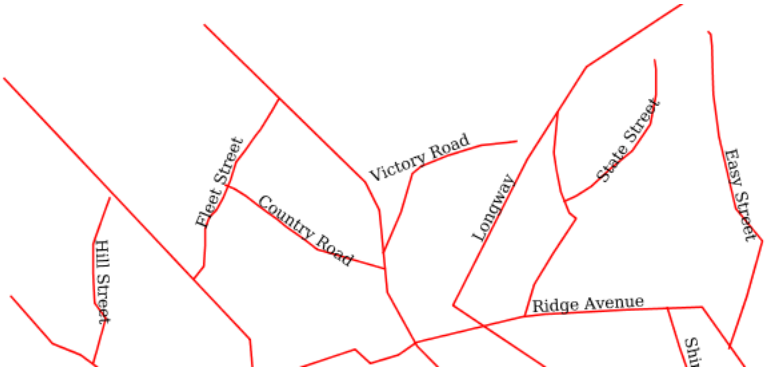


Fig. 6.131: *Line with default label*

Code

```
* {
  stroke: red;
  label: [name];
  label-offset: 7px;
  font-fill: black;
}
```

Details

This example is line by line identical to the previous one, but it add a new attribute "label-offset", which in the case of lines, when having a single value, is intepreted as a perpendicular offset from the line. The label is painted along a straight line, parallel to the line orientation in the center point of the label.

Label following line

This example renders the text label to follow the contour of the lines.

Code

```
* {
  stroke: red;
  label: [name];
  font-fill: black;
  label-follow-line: true;
}
```

Fig. 6.132: *Label following line*

Details

As the *Line with default label* example showed, the default label behavior isn't optimal.

This example is similar to the *Line with default label* example with the exception of **line 5** where the "label-follow-line" option is specified, which forces the labels to strickly follow the line.

Not all labels are visible partly because of conflict resolution, and partly because the renderer cannot find a line segment long and "straight" enough to paint the label (labels are not painted over sharp turns by default).

Optimized label placement

This example optimizes label placement for lines such that the maximum number of labels are displayed.

Code

```
* {
  stroke: red;
  label: [name];
  font-fill: black;
  label-follow-line: true;
  label-max-angle-delta: 90;
  label-max-displacement: 400;
  label-repeat: 150;
}
```

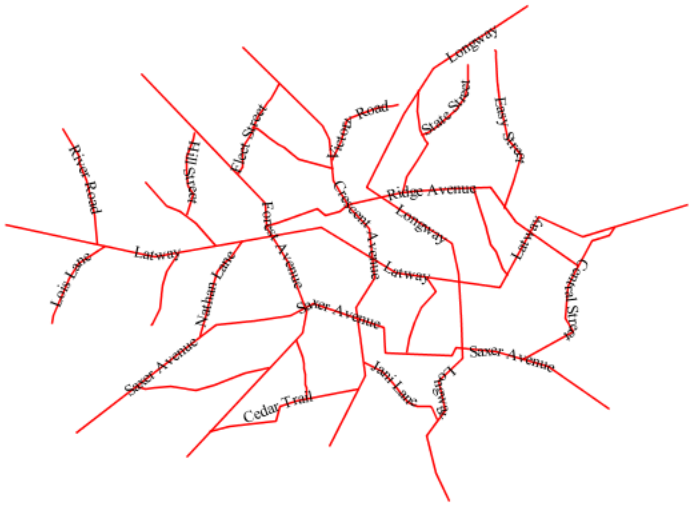


Fig. 6.133: *Optimized label*

Details

This example is similar to the previous example, *Label following line*. The only differences are contained in **lines 6-8**. **Line 6** sets the maximum angle that the label will follow. This sets the label to never bend more than 90 degrees to prevent the label from becoming illegible due to a pronounced curve or angle. **Line 7** sets the maximum displacement of the label to be 400 pixels. In order to resolve conflicts with overlapping labels, GeoServer will attempt to move the labels such that they are no longer overlapping. This value sets how far the label can be moved relative to its original placement. Finally, **line 8** sets the labels to be repeated every 150 pixels. A feature will typically receive only one label, but this can cause confusion for long lines. Setting the label to repeat ensures that the line is always labeled locally.

Optimized and styled label

This example improves the style of the labels from the *Optimized label placement* example.

Code

```
1
2  * {
3    stroke: red;
4    label: [name];
5    font-family: Arial;
6    font-weight: bold;
     font-fill: black;
```

Fig. 6.134: *Optimized and styled label*

```
  font-size: 10;
  halo-color: white;
  halo-radius: 1;
  label-follow-line: true;
  label-max-angle-delta: 90;
  label-max-displacement: 400;
  label-repeat: 150;
}
```

Details

This example is similar to the *Optimized label placement*. The only differences are:

- The font family and weight have been specified
- In order to make the labels easier to read, a white "halo" has been added. The halo draws a thin 1 pixel white border around the text, making it stand out from the background.

Attribute-based line

This example styles the lines differently based on the "type" (Road class) attribute.

Code

```
[type = 'local-road'] {
  stroke: #009933;
  stroke-width: 2;
  z-index: 0;
}

[type = 'secondary'] {
  stroke: #0055CC;
  stroke-width: 3;
  z-index: 1;
```

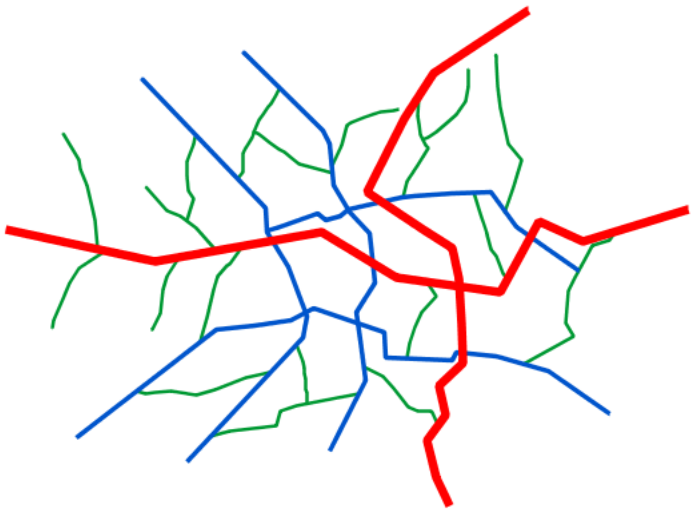


Fig. 6.135: *Attribute-based line*

```
11                                  }
12
13                                  [type = 'highway'] {
14                                    stroke: #FF0000;
15                                    stroke-width: 6;
16                                    z-index: 2;
17                                  }
```

Details

Note: Refer to the *Example lines layer* to see the attributes for the layer. This example has eschewed labels in order to simplify the style, but you can refer to the example *Optimized and styled label* to see which attributes correspond to which points.

There are three types of road classes in our fictional country, ranging from back roads to high-speed freeways: "highway", "secondary", and "local-road". In order to make sure the roads are rendered in the proper order of importance, a "z-index" attribute has been placed in each rule.

The three rules are designed as follows:

Rule order	Rule name / type	Color	Size
1	local-road	`#009933` (green)	2
2	secondary	`#0055CC` (blue)	3
3	highway	`#FF0000` (red)	6

Lines 1-5 comprise the first rule, the filter matches all roads that the

"type" attribute has a value of "local-road". If this condition is true for a particular line, the rule renders it dark green, 2 pixels wide. All these lines are rendered first, and thus sit at the bottom of the final map.

Lines 7-11 match the "secondary" roads, painting them dark blue, 3 pixels wide. Given the "z-index" is 1, they are rendered after the local roads, but below the highways.

Lines 13-17 match the "highway" roads, painting them red 6 pixels wide. These roads are pained last, thus, on top of all others.

Zoom-based line

This example alters the *Simple line* style at different zoom levels.

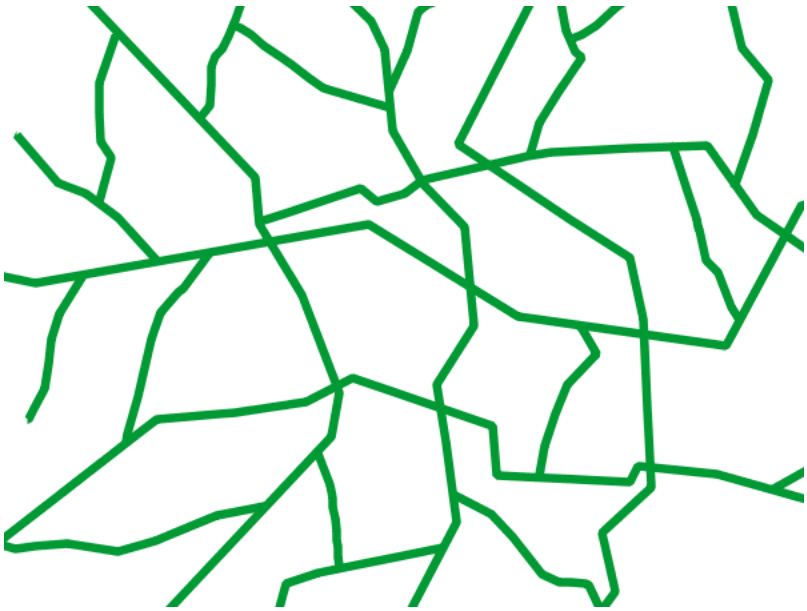


Fig. 6.136: *Zoom-based line: Zoomed in*

Code

```
* {
  stroke: #009933;
}

[@sd < 180M] {
  stroke-width: 6;
}

[@sd > 180M] [@sd < 360M] {
  stroke-width: 4;
}

[@sd > 360M] {
```

Fig. 6.137: *Zoom-based line: Partially zoomed*

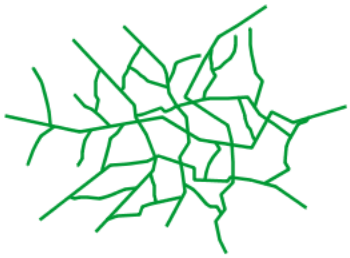


Fig. 6.138: *Zoom-based line: Zoomed out*

```
14                                  stroke-width: 2;
15                                }
```

Details

It is often desirable to make shapes larger at higher zoom levels when creating a natural-looking map. This example varies the thickness of the lines according to the zoom level (or more accurately, scale denominator). Scale denominators refer to the scale of the map. A scale denominator of 10,000 means the map has a scale of 1:10,000 in the units of the map projection.

Note: Determining the appropriate scale denominators (zoom levels) to use is beyond the scope of this example.

This style contains three rules. The three rules are designed as follows:

Rule order	Rule name	Scale denominator	Line width
1	Large	1:180,000,000 or less	6
2	Medium	1:180,000,000 to 1:360,000,000	4
3	Small	Greater than 1:360,000,000	2

The order of these rules does not matter since the scales denominated in each rule do not overlap.

The first rule provides the stroke color used at all zoom levels, dark gray, while the other three rules cascade over it applying the different stroke widths based on the current zoom level leveraging the "@sd" pseudo attribute. The "@sd" pseudo attribute can only be compared using the "<" and ">" operators, using any other operator will result in errors.

The result of this style is that lines are drawn with larger widths as one zooms in and smaller widths as one zooms out.

Polygons

Polygons are two dimensional shapes that contain both an outer edge (or "stroke") and an inside (or "fill"). A polygon can be thought of as an irregularly-shaped point and is styled in similar ways to points.

Example polygons layer

The `polygons layer` used below contains county information for a fictional country. For reference, the attribute table for the polygons is included below.

fid (Feature ID)	**name** (County name)	**pop** (Population)
polygon.1	Irony County	412234
polygon.2	Tracker County	235421
polygon.3	Dracula County	135022
polygon.4	Poly County	1567879
polygon.5	Bearing County	201989
polygon.6	Monte Cristo County	152734
polygon.7	Massive County	67123
polygon.8	Rhombus County	198029

`Download the polygons shapefile`

Simple polygon

This example shows a polygon filled in blue.

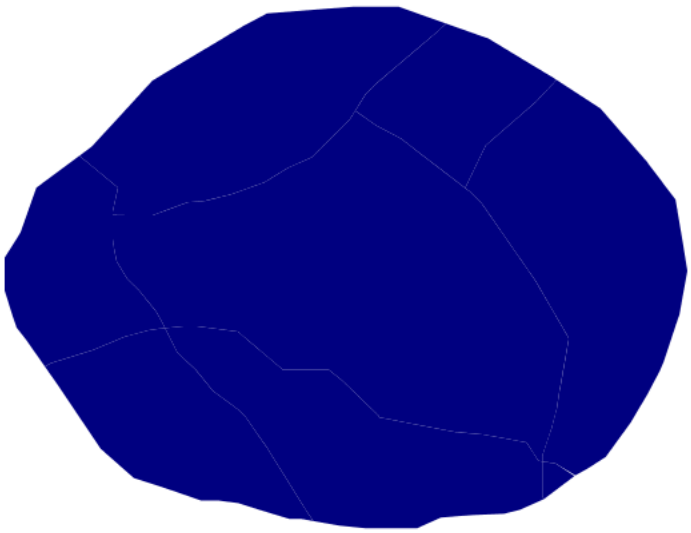


Fig. 6.139: *Simple polygon*

Code

```
* {
  fill: #000080;
}
```

Details

This simple rule applies a dark blue (`#000080`) fill to all the polygons in the dataset.

Note: The light-colored borders around the polygons in the figure are artifacts of the renderer caused by the polygons being adjacent. There is no border in this style.

Simple polygon with stroke

This example adds a 2 pixel white stroke to the *Simple polygon* example.

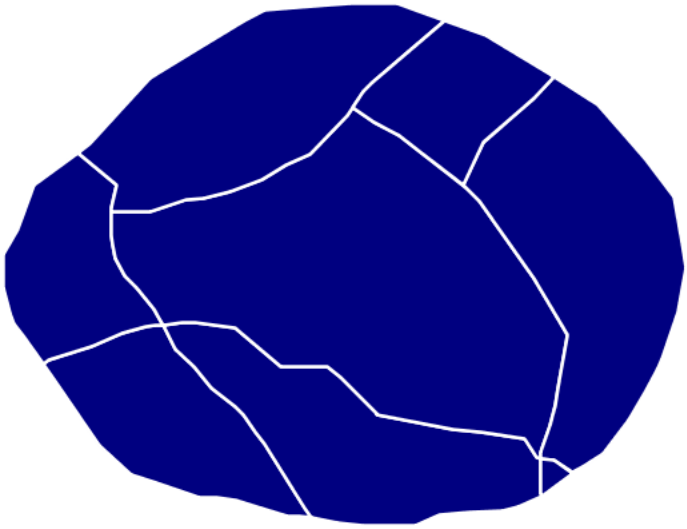


Fig. 6.140: *Simple polygon with stroke*

Code

```
* {
  fill: #000080;
  stroke: #FFFFFF;
  stroke-width: 2;
}
```

Details

This example is similar to the *Simple polygon* example above, with the addition of the "stroke" and "stroke-width" attributes, that add a white, 2 pixels wide border around each polygon.

Transparent polygon

This example builds on the *Simple polygon with stroke* example and makes the fill partially transparent by setting the opacity to 50%.

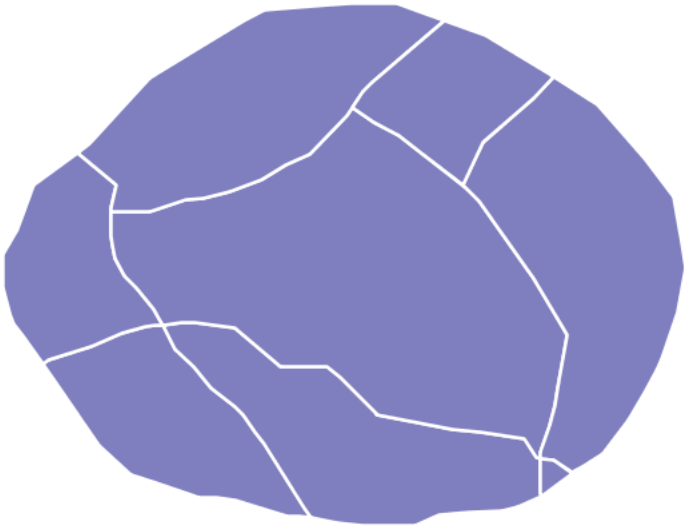


Fig. 6.141: *Transparent polygon*

Code

```
1  * {
2    fill: #000080;
3    fill-opacity: 0.5;
4    stroke: #FFFFFF;
5    stroke-width: 2;
6  }
```

Details

This example is similar to the *Simple polygon with stroke* example, save for defining the fill's opacity in **line 3**. The value of 0.5 results in partially transparent fill that is 50% opaque. An opacity value of 1 would draw the fill as 100% opaque, while an opacity value of 0 would result in a completely transparent (0% opaque) fill. In this example, since the background is white, the dark blue looks lighter. Were the points imposed on a dark background, the resulting color would be darker.

Graphic fill

This example fills the polygons with a tiled graphic.

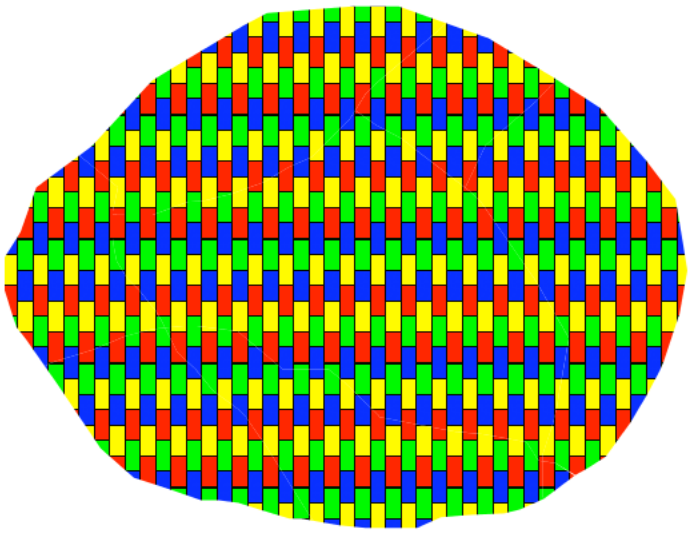


Fig. 6.142: *Graphic fill*

Code

```
1  * {
2    fill: url("colorblocks1.png");
3    fill-mime: 'image/png';
4  }
```

Details

This style fills the polygon with a tiled graphic. The graphic is selected providing a url for the fill, which in this case is meant to the relative to the `styles` directory contained within the data directory (an absolute path could have been provided, as well as a internet reference). **Line 3** specifies that the image itself is a png (by default the code assumes jpegs are used and will fail to parse the file unless we specify its mime type). The size of the image is not specified, meaning the native size is going to be used. In case a rescale is desired, the "fill-size" attribute can be used to force a different size.

Fig. 6.143: *Graphic used for fill*

Hatching fill

This example fills the polygons with a hatching pattern.

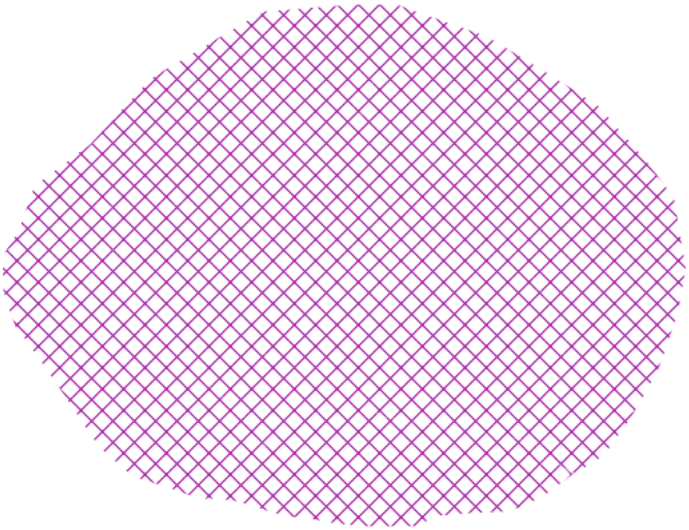


Fig. 6.144: *Hatching fill*

Code

```
1  * {
2    fill: symbol("shape://times");
3    :fill {
4      size: 16;
5      stroke: #990099;
6      stroke-width: 1px;
7    }
8  }
```

Details

In this example the fill is specified to be the "shape://times" symbol, which is going to be tiled creating a cross-hatch effect.

The details of the hatch are specified at **line 3***, where the pseudo-selector ":fill" is used to match the contents of the fill, and specify that we want a symbol large 16 pixels (the larger the symbol, the coarser the cross hatch will be), and painted with a 1 pixel wide purple stroke.

Polygon with default label

This example shows a text label on the polygon. In the absence of any other customization, this is how a label will be displayed.

Fig. 6.145: *Polygon with default label*

Code

```
* {
  fill: #40FF40;
  stroke: white;
  stroke-width: 2;
  label: [name];
  font-fill: black;
}
```

Details

The single rule in the CSS applies to all feature: first it fills all polygons a light green with white outline, and thn applies the "name" attribute as the label, using the default font (Times), with black color and default font size (10 px).

Label halo

This example alters the look of the *Polygon with default label* by adding a white halo to the label.

Fig. 6.146: *Label halo*

Code

```
1  * {
2    fill: #40FF40;
3    stroke: white;
4    stroke-width: 2;
5    label: [name];
6    font-fill: black;
7    halo-color: white;
8    halo-radius: 3;
9  }
```

Details

This example builds on *Polygon with default label*, with the addition of a halo around the labels on **lines 7-8**. A halo creates a color buffer around the label to improve label legibility. **Line 9** sets the radius of the halo, extending the halo 3 pixels around the edge of the label, and **line 8** sets the color of the halo to white. Since halos are most useful when set to a sharp contrast relative to the text color, this example uses a white halo around black text to ensure optimum readability.

Polygon with styled label

This example improves the label style from the *Polygon with default label* example by centering the label on the polygon, specifying a different font name and size, and setting additional label placement optimizations.

Fig. 6.147: *Polygon with styled label*

Code

```
* {
  fill: #40FF40;
  stroke: white;
  stroke-width: 2;
  label: [name];
  font-family: Arial;
  font-size: 11px;
  font-style: normal;
  font-weight: bold;
  font-fill: black;
  label-anchor: 0.5 0.5;
  label-auto-wrap: 60;
  label-max-displacement: 150;
}
```

Details

This example is similar to the *Polygon with default label* example, with additional styling options for the labels.

The font is setup to be Arial, 11 pixels, "normal" (as opposed to "italic") and bold.

The "label-anchor" affects where the label is placed relative to the centroid of the polygon, centering the label by positioning it 50% (or 0.5) of the way horizontally along the centroid of the polygon, as well as vertically in exactly the same way.

Finally, there are two added touches for label placement optimization: The "label-auto-wrap" attribute ensures that long labels are

split across multiple lines by setting line wrapping on the labels to 60 pixels, whilst the "label-max-displacement" allows the label to be displaced by up to 150 pixels. This ensures that labels are compacted and less likely to spill over polygon boundaries. Notice little Massive County in the corner, whose label is now displayed.

Attribute-based polygon

This example styles the polygons differently based on the "pop" (Population) attribute.

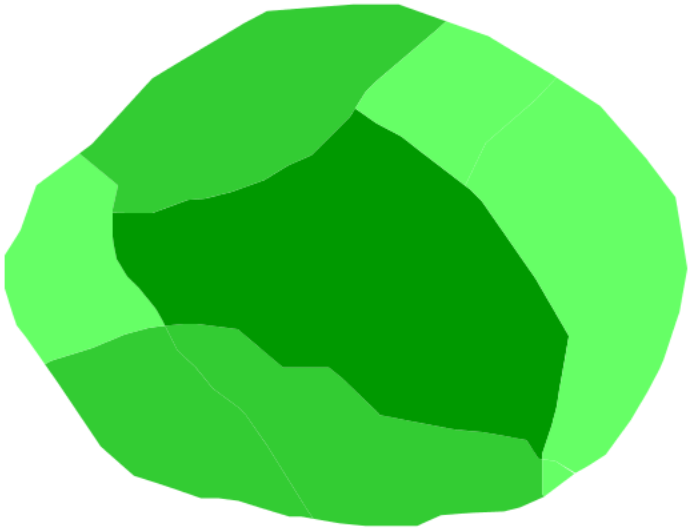


Fig. 6.148: *Attribute-based polygon*

Code

```
[parseLong(pop) < 200000] {
  fill: #66FF66;
}

[parseLong(pop) >= 200000] [parseLong(pop) < 500000] {
  fill: #33CC33;
}

[parseLong(pop) >= 500000] {
  fill: #009900;
}
```

Details

Note: Refer to the *Example polygons layer* to see the attributes for the layer. This example has eschewed labels in order to simplify the style, but you can refer to the example *Polygon with styled label* to see which attributes correspond to which polygons.

Each polygon in our fictional country has a population that is represented by the population ("pop") attribute. This style contains three rules that alter the fill based on the value of "pop" attribute, with smaller values yielding a lighter color and larger values yielding a darker color.

The three rules are designed as follows:

Rule order	**Rule name**	**Population** ("pop")	**Color**
1	SmallPop	Less than 200,000	`#66FF66`
2	MediumPop	200,000 to 500,000	`#33CC33`
3	LargePop	Greater than 500,000	`#009900`

The order of the rules does not matter in this case, since each shape is only rendered by a single rule.

The first rule fills light green all polygons whose "pop" attribute is below 200,000, the second paints medium green all poygons whose "pop" attribute is between 200,000 and 500,000, while the third rule paints dark green the remaining polygons.

What's interesting in the filters is the use of the "parseLong" filter function: this function is necessary because the "pop" attribute is a string, leaving it as is we would have a string comparison, whilst the function turns it into a number, ensuring proper numeric comparisons instead.

Zoom-based polygon

This example alters the style of the polygon at different zoom levels.

Code

```
* {
  fill: #0000CC;
  stroke: black;
}

[@sd < 100M] {
   stroke-width: 7;
   label: [name];
   label-anchor: 0.5 0.5;
   font-fill: white;
   font-family: Arial;
   font-size: 14;
   font-weight: bold;
}
```

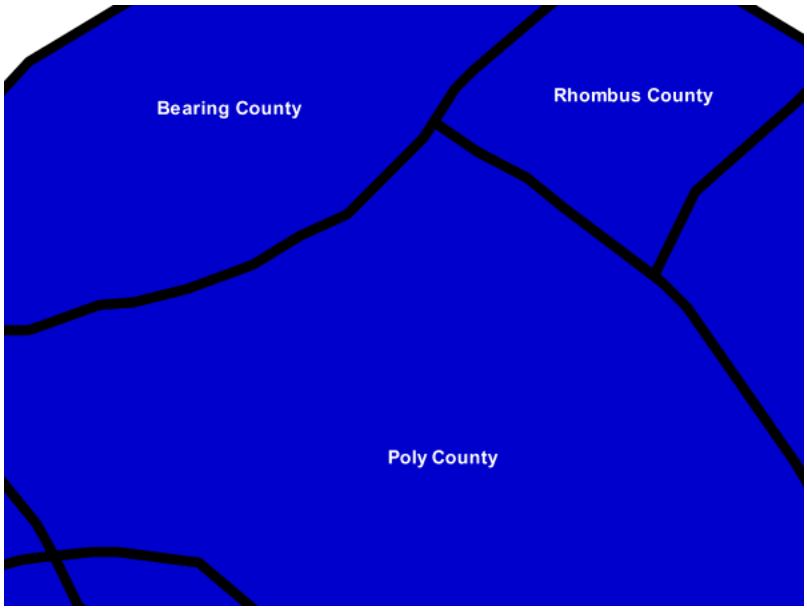


Fig. 6.149: *Zoom-based polygon: Zoomed in*

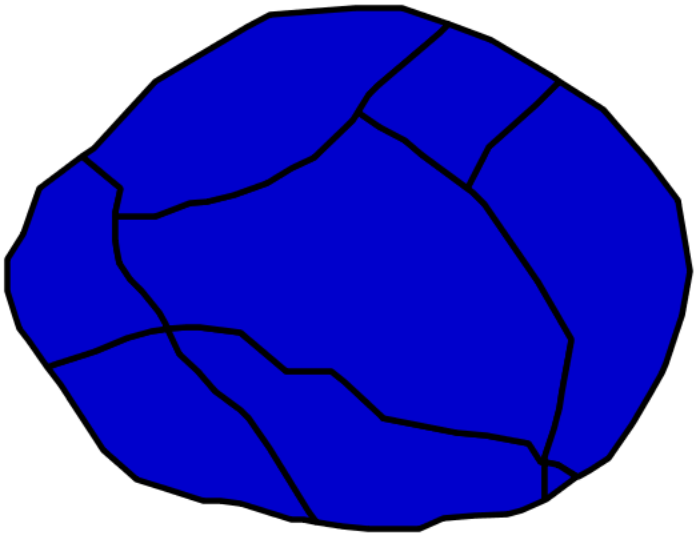


Fig. 6.150: *Zoom-based polygon: Partially zoomed*

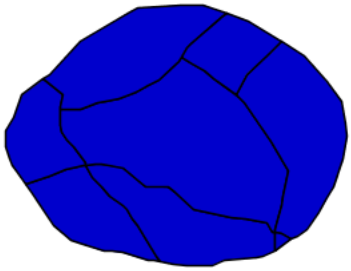


Fig. 6.151: *Zoom-based polygon: Zoomed out*

```
15
16                              [@sd > 100M] [@sd < 200M] {
17                                 stroke-width: 4;
18                              }
19
20                              [@sd > 200M] {
21                                 stroke-width: 1;
22                              }
```

Details

It is often desirable to make shapes larger at higher zoom levels when creating a natural-looking map. This example varies the thickness of the lines according to the zoom level. Polygons already do this by nature of being two dimensional, but another way to adjust styling of polygons based on zoom level is to adjust the thickness of the stroke (to be larger as the map is zoomed in) or to limit labels to only certain zoom levels. This is ensures that the size and quantity of strokes and labels remains legible and doesn't overshadow the polygons themselves.

Zoom levels (or more accurately, scale denominators) refer to the scale of the map. A scale denominator of 10,000 means the map has a scale of 1:10,000 in the units of the map projection.

Note: Determining the appropriate scale denominators (zoom levels) to use is beyond the scope of this example.

This style contains three rules, defined as follows:

Rule order	Rule name	Scale denominator	Stroke width	Label display?
1	Large	1:100,000,000 or less	7	Yes
2	Medium	1:100,000,000 to 1:200,000,000	4	No
3	Small	Greater than 1:200,000,000	2	No

The first rule (**lines 1-4**) defines the attributes that are not scale dependent: dark blue fill, black outline.

The second (**lines 6-14**) rule provides specific overrides for the higher zoom levels, asking for a large stroke (7 pixels) and a label, which is only visible at this zoom level. The label is white, bold, Arial 14 pixels, its contents are coming form the "name" attribute.

The third rule (**lines 16-18**) specifies a stroke width of 4 pixels for medium zoom levels, whilst for low zoom levels the stroke width is set to 1 pixel by the last rule (**lines 20-22**).

The resulting style produces a polygon stroke that gets larger as one zooms in and labels that only display when zoomed in to a sufficient level.

Rasters

Rasters are geographic data displayed in a grid. They are similar to image files such as PNG files, except that instead of each point containing visual information, each point contains geographic information in numerical form. Rasters can be thought of as a georeferenced table of numerical values.

One example of a raster is a Digital Elevation Model (DEM) layer, which has elevation data encoded numerically at each georeferenced data point.

Example raster

The `raster layer` that is used in the examples below contains elevation data for a fictional world. The data is stored in EPSG:4326 (longitude/latitude) and has a data range from 70 to 256. If rendered in grayscale, where minimum values are colored black and maximum values are colored white, the raster would look like this:

`Download the raster file`

Two-color gradient

This example shows a two-color style with green at lower elevations and brown at higher elevations.

Fig. 6.152: *Raster file as rendered in grayscale*

Fig. 6.153: *Two-color gradient*

Code

```
* {
  raster-channels: auto;
  raster-color-map:
                    color-map-entry(#008000, 70)
                    color-map-entry(#663333, 256);
}
```

Details

There is a single rule which applies a color map to the raster data.

The "raster-channels" attribute activates raster symbolization, the "auto" value is indicates that we are going to use the default choice of bands to symbolize the output (either gray or RBG/RGBA depending on the input data). There is also the possibility of providing a band name or a list of band names in case we want to choose specific bands out of a multiband input, e.g., "1" or "1 3 7".

The "raster-color-map" attribute builds a smooth gradient between two colors corresponding to two elevation values. Each "color-map-entry" represents one entry or anchor in the gradient:

- The first argument is the color

- The second argument is the value at which we anchor the color
- An optional third argument could specify the opacity of the pixels, as a value between 0 (fully transparent) and 1 (fully opaque). The default, when not specified, is 1, fully opaque.

  Line 4 sets the lower value of 70, which is styled a opaque dark green (`#008000`), and **line 5** sets the upper value of 256, which is styled a opaque dark brown (`#663333`). All data values in between these two quantities will be linearly interpolated: a value of 163 (the midpoint between 70 and 256) will be colored as the midpoint between the two colors (in this case approximately `#335717`, a muddy green).

Transparent gradient

This example creates the same two-color gradient as in the *Two-color gradient* as in the example above but makes the entire layer mostly transparent by setting a 30% opacity.

Fig. 6.154: *Transparent gradient*

Code

```
1  * {
2    raster-channels: auto;
3    raster-opacity: 0.3;
4    raster-color-map: color-map-entry(#008000, 70)
5                      color-map-entry(#663333, 256);
6  }
```

Details

This example is similar to the *Two-color gradient* example save for the addition of **line 3**, which sets the opacity of the layer to 0.3 (or 30% opaque). An opacity value of 1 means that the shape is drawn 100% opaque, while an opacity value of 0 means that the shape is rendered as completely transparent. The value of 0.3 means that the the raster partially takes on the color and style of whatever is

drawn beneath it. Since the background is white in this example, the colors generated from the "raster-color-map" look lighter, but were the raster imposed on a dark background the resulting colors would be darker.

Brightness and contrast

This example normalizes the color output and then increases the brightness by a factor of 2.

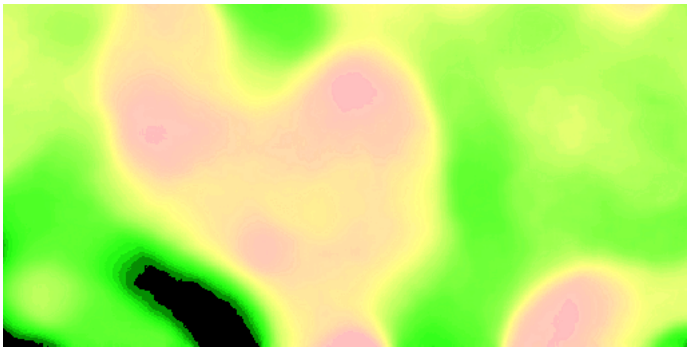


Fig. 6.155: *Brightness and contrast*

Code

```
1  * {
2    raster-channels: auto;
3    raster-contrast-enhancement: normalize;
4    raster-gamma: 0.5;
5    raster-color-map: color-map-entry(#008000, 70)
6                      color-map-entry(#663333, 256);
7  }
```

Details

This example is similar to the *Two-color gradient*, save for the addition of the contrast enhancement and gamma attributes on **lines 3-4**. **Line 3** normalizes the output by increasing the contrast to its maximum extent. **Line 4** then adjusts the brightness by a factor of 0.5. Since values less than 1 make the output brighter, a value of 0.5 makes the output twice as bright.

Three-color gradient

This example creates a three-color gradient in primary colors. In addition, we want to avoid displaying data outside of the chosen range, leading some data not to be rendered at all.

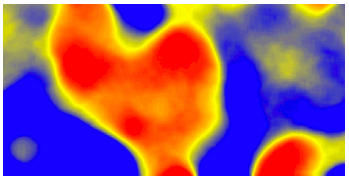


Fig. 6.156: *Three-color gradient*

Code

```
1  * {
2    raster-channels: auto;
3    raster-color-map:
4                      color-map-entry(black, 150, 0)
5                      color-map-entry(blue, 150)
6                      color-map-entry(yellow, 200)
7                      color-map-entry(red, 250)
8                      color-map-entry(black, 250, 0)
9  }
```

Details

This example creates a three-color gradient, with two extra rules to make ranges of color disappear. The color map behavior is such that any value below the lowest entry gets the same color as that entry, and any value above the last entry gets the same color as the last entry, while everything in between is linearly interpolated (all values must be provided from lower to higher). **Line 4** associates value 150 and below with a transparent color (0 opacity, that is, fully transparent), and so does **line 8**, which makes transparent every value above 250. The lines in the middle create a gradient going from blue, to yellow, to red.

Alpha channel

This example creates an "alpha channel" effect such that higher values are increasingly transparent.

Code

```
1  * {
2    raster-channels: auto;
3    raster-color-map: color-map-entry(#008000, 70)
4                      ␣
↪                        color-map-entry(#663333, 256, 0);
5  }
```

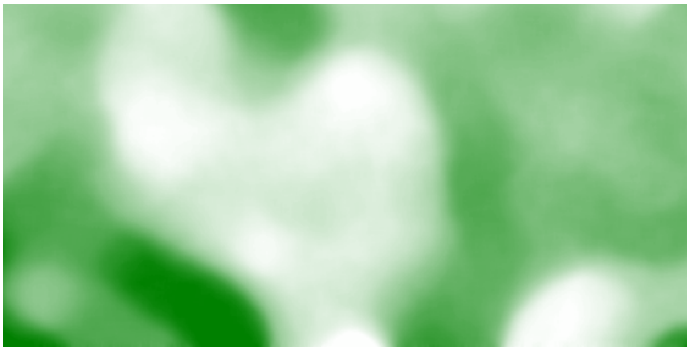


Fig. 6.157: *Alpha channel*

Details

An alpha channel is another way of referring to variable transparency. Much like how a gradient maps values to colors, each entry in a "raster-color-map" can have a value for opacity (with the default being 1.0 or completely opaque).

In this example, there is a "raster-color-map" with two entries: **line 3** specifies the lower bound of 70 be colored dark green (`#008000`), while **line 4** specifies the upper bound of 256 also be colored dark green but with an opacity value of 0. This means that values of 256 will be rendered at 0% opacity (entirely transparent). Just like the gradient color, the opacity is also linearly interpolated such that a value of 163 (the midpoint between 70 and 256) is rendered at 50% opacity.

Discrete colors

This example shows a gradient that is not linearly interpolated but instead has values mapped precisely to one of three specific colors.

Fig. 6.158: *Discrete colors*

Code

```
1  * {
2    raster-channels: auto;
3    raster-color-map-type: intervals;
4    raster-color-map: color-map-entry(#008000, 150)
5                      color-map-entry(#663333, 256);
6  }
```

Details

Sometimes color bands in discrete steps are more appropriate than a color gradient. The "raster-color-map-type: intervals" attribute sets the display to output discrete colors instead of a gradient. The values in each entry correspond to the upper bound for the color band such that colors are mapped to values less than the value of one entry but greater than or equal to the next lower entry. For example, **line 4** colors all values less than 150 to dark green (`#008000`) and **line 5** colors all values less than 256 but greater than or equal to 150 to dark brown (`#663333`).

Many color gradient

This example shows a gradient interpolated across eight different colors.

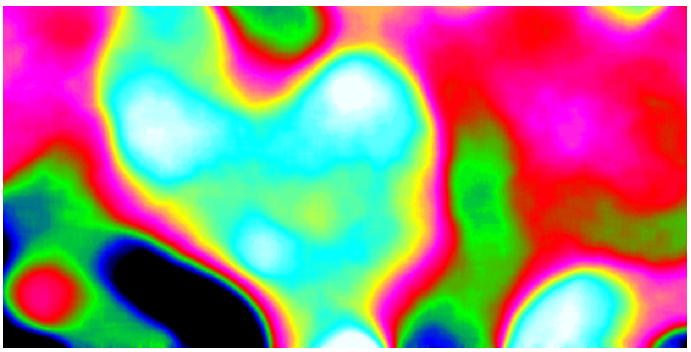


Fig. 6.159: *Many color gradient*

Code

```
1  * {
2    raster-channels: auto;
3    raster-color-map:
4            color-map-entry(black, 95)
5            color-map-entry(blue, 110)
6            color-map-entry(green, 135)
7            color-map-entry(red, 160)
8            color-map-entry(purple, 185)
```

```
                color-map-entry(yellow, 210)
                color-map-entry(cyan, 235)
                color-map-entry(white, 256)
        }
```

Details

This example is similar to the previous ones, and creates a color gradient between 8 colors as reported in the following table

Entry number	Value	Color
1	95	Black
2	110	Blue
3	135	Green
4	160	Red
5	185	Purple
6	210	Yellow
7	235	Cyan
8	256	White

6.4.14 Styling examples

The following pages contain CSS styling examples grouped by specific topics.

Fills with randomized symbols

It is possible to generate fills by randomly repeating a symbol in the polygons to be filled. Please refer to the *equivalent SLD chapter* for details on the meaning of the various options.

Simple random distribution

Here is an example distributing up to 50 small "slash" symbols in a 100x100 pixel tile (in case of conflicts the symbol will be skipped), enabling random symbol rotation), and setting the seed to "5" to get a distribution different than the default one:

```
* {
  fill: symbol("shape://slash");
  :fill {
    size: 8;
    stroke: blue;
    stroke-width: 4;
    stroke-linecap: round;
  };
  stroke: black;
  fill-random: grid;
  fill-random-seed: 5;
  fill-random-rotation: free;
```

```
    fill-random-symbol-count: 50;
    fill-random-tile-size: 100;
}
```

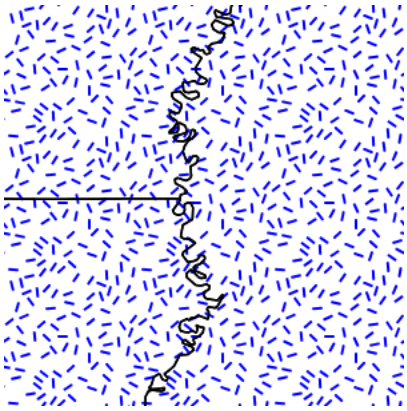


Fig. 6.160: *Random distribution of a diagonal line*

Thematic map using point density

Randomized distributions can also be used for thematic mapping, for example, here is the SLD for a version of topp:states that displays the number of inhabitantìs varying the density of a random point distribution:

```
* {
  fill: symbol("circle");
  stroke: black;
  fill-random: grid;
  fill-random-tile-size: 100;

  :fill {
    size: 2;
    fill: darkgray;
  };
  /* @title low */
  [PERSONS < 2000000] {
    fill-random-symbol-count: 50;
  };
  /* @title mid */
  [PERSONS >= 2000000] [PERSONS < 4000000] {
    fill-random-symbol-count: 150;
  };
  /* @title high */
  [PERSONS >= 4000000] {
    fill-random-symbol-count: 500;
  }
}
```

Fig. 6.161: *Thematic map via point density approach*

Using transformation functions

The transformation functions supported described in SLD and described in the *equivalent SLD chapter* are also available in CSS, the following shows examples of how they can be used.

Recode

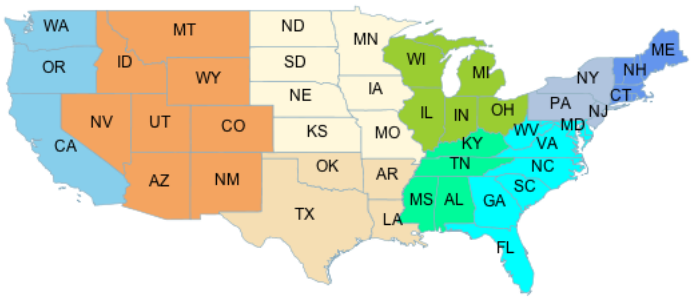
The `Recode` filter function transforms a set of discrete values for an attribute into another set of values, by applying a *(input, output)* mapping onto the values of the variable/expression that is provided as the first input of the function.

Consider a chloropleth map of the US states dataset using the fill color to indicate the topographic regions for the states. The dataset has an attribute `SUB_REGION` containing the region code for each state. The `Recode` function is used to map each region code into a different color.

Note: It is to be noted that the following example specifies colors as hex string as opposed to native CSS color names, this is because the function syntax is expressed in CQL, which does not have support for native CSS color names.

```
* {
  fill: [recode(strTrim(SUB_REGION),
          'N Eng', '#6495ED',
          'Mid Atl', '#B0C4DE',
          'S Atl', '#00FFFF',
          'E N Cen', '#9ACD32',
          'E S Cen', '#00FA9A',
          'W N Cen', '#FFF8DC',
          'W S Cen', '#F5DEB3',
          'Mtn', '#F4A460',
          'Pacific', '#87CEEB')];
  stroke: lightgrey;
  label: [STATE_ABBR];
  font-family: 'Arial';
  font-fill: black;
```

```
  label-anchor: 0.5 0.5;
}
```

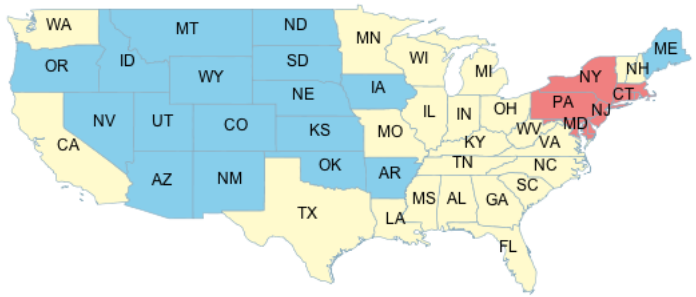


Categorize

The `Categorize` filter function transforms a continuous-valued attribute into a set of discrete values by assiging ranges of values and turning them into a color, size, width, opacity, etc.

In the following example a coropleth map is build associating a color to the state population density in the ranges [<= 20], [20 - 100], and [> 100].

```
* {
  fill: [categorize(
          PERSONS / LAND_KM,
          '#87CEEB',
          20,
          '#FFFACD',
          100,
          '#F08080')];
  stroke : lightgrey;
  label: [STATE_ABBR];
  font-family: 'Arial';
  font-fill: black;
  label-anchor: 0.5 0.5;
}
```



Interpolate

The `Interpolate` filter function transforms a continuous-valued attribute into another continuous range of values by applying piecewise interpolation.

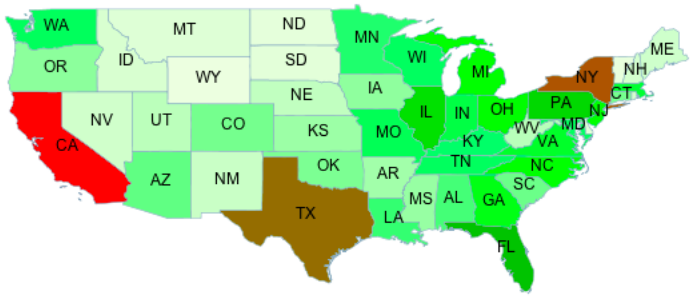
The result will work for numeric values such as size, width, opacity when operating in `numeric` **interpolation method** (the default), and for colors when working in `color` mode.

The type of curve fitting the specified points can be either `linear` (the default), `cubic` or `cosine`, these values are known as the **interpolation mode**.

Both the interpolation method and mode are optional, and if provided, they are added at the end of the input list.

In the following example the state population is mapped to a continuous color scale in a rather compact way using the interpolate function:

```
* {
  fill: [Interpolate(
          PERSONS,
          0, '#FEFEEE',
          9000000, '#00FF00',
          23000000, '#FF0000',
          'color',
          'linear')];
  stroke : lightgrey;
  label: [STATE_ABBR];
  font-family: 'Arial';
  font-fill: black;
  label-anchor: 0.5 0.5;
}
```



KML

Detecting raster to vector switch in KML

GeoServer 2.4 added a new icon server that KML output uses to make sure the point symbolisers look the same as in a normal WMS call no matter what scale they are looked at.

This may pose some issue when working in the default KML generation mode, where the map is a ground overlay up to a certain scale, and switches to a vector, clickable representation once the number of features in the visualization fall below a certain scale (as controlled by the `KMSCORE` parameter): the end user is not informed "visually" that the switch happened.

There is however a custom enviroment variable, set by the KML generator, that styles can leverage to know whether the KML generation is happening in ground overlay or vector mode.

The following example leverages this function to show a larger point symbol when points become clickable:

```
* {
  mark: symbol("circle");
}

:mark [env('kmlOutputMode') = 'vector'] {
  size: 8;
}

:mark {
  size: 4;
  fill: yellow;
  stroke: black;
}
```

This will result in the following output:

Fig. 6.162: *Raster output, points are not yet clickable*

One important bit about the above CSS is that the order of the rules is important. The CSS to SLD translator uses specificity to decide which rule overrides which other one, and the specificity is driven, at the time of writing, only by scale rules and access to attributes. The filter using the `kmlOutputMode` filter is not actually using any feature attribute, so it has the same specificity as the catch all `:mark` rule. Putting it first ensures that it overrides the catch all rule anyways, while putting it second would result in the output size being always 4.

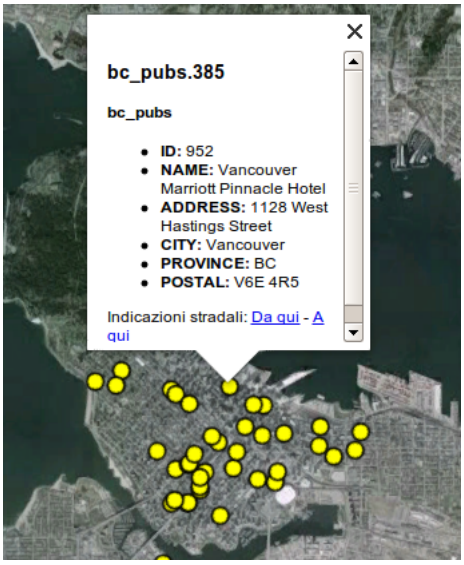


Fig. 6.163: *Vector output, points are clickable and painted as larger icons*

Getting KML marks similar to the old KML encoder

The old KML generator (prior to GeoServer 2.4) was not able to truly respect the marks own shape, and as a result, was simply applying the mark color to a fixed bull's eye like icon, for example:

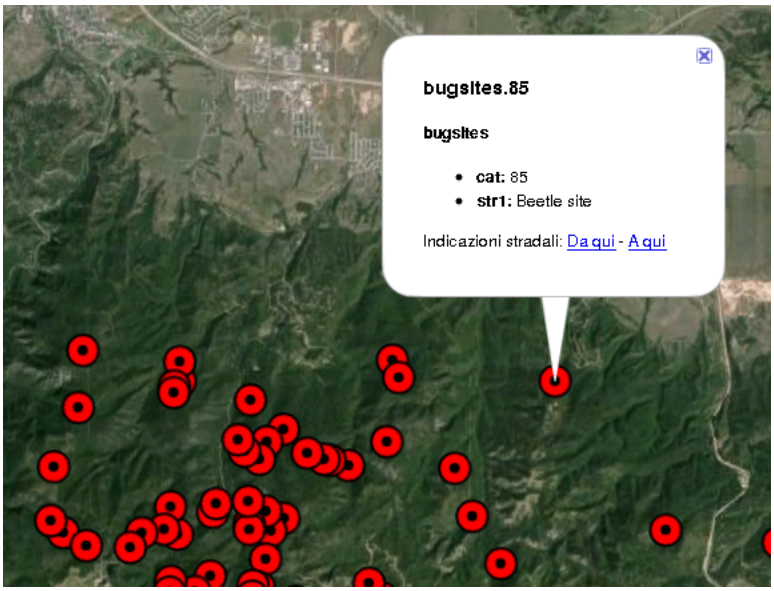
Starting with GeoServer 2.4 the KML engine has been rewritten, and among other things, it can produce an exact representation of the marks, respecting not only color, but also shape and stroking. However, what if one want to reproduce the old output look?

The solution is to leverage the ability to respect marks appearance to the letter, and combine two superimposed marks to generate the desired output:

```
* {
  mark: symbol('circle'), symbol('circle');
  mark-size: 12, 4;
}

:nth-mark(1) {
  fill: red;
  stroke: black;
  stroke-width: 2;
```

```
}

:nth-mark(2) {
  fill: black;
}
```

Which results in the following Google Earth output:



Miscellaneous

Markers sized by an attribute value

The following produces square markers at each point, but these are sized such that the area of each marker is proportional to the `REPORTS` attribute. When zoomed in (when there are less points in view) the size of the markers is doubled to make the smaller points more noticeable.

```
* {
  mark: symbol(square);
}

[@sd > 1M] :mark {
  size: [sqrt(REPORTS)];
}

/* So␣
↪that single-report points can be more easily seen */
[@sd < 1M] :mark {
  size: [sqrt(REPORTS)*2];
}
```

This example uses the `sqrt` function. There are many functions available for use in CSS and SLD. For more details read - *Filter Func-*

tion Reference

Specifying a geometry attribute

In some cases, typically when using a database table with multiple geometry columns, it's necessary to specify which geometry to use. For example, let's suppose you have a table containing routes `start` and `end` both containing point geometries. The following CSS will style the start with a triangle mark, and the end with a square.

```
* {
    geometry: [start],           [end];
    mark:     symbol(triangle), symbol(square);
}
```

Generating a geometry (Geometry Transformations)

Taking the previous example a bit further, we can also perform computations on-the-fly to generate the geometries that will be drawn. Any operation that is available for GeoServer *Geometry transformations in SLD* is also available in CSS styles. To use them, we simply provide a more complex expression in the `geometry` property. For example, we could mark the start and end points of all the paths in a line layer (you can test this example out with any line layer, such as the `sf:streams` layer that is included in GeoServer's default data directory.)

```
* {
    geometry:␣
↪[startPoint(the_geom)], [endPoint(the_geom)];
    mark:     symbol(triangle),       symbol(square);
}
```

Rendering different geometry types (lines/points) with a single style

As one more riff on the geometry examples, we'll show how to render both the original line and the start/endpoints in a single style. This is accomplished by using `stroke-geometry` and `mark-geometry` to specify that different geometry expressions should be used for symbols compared with strokes.

```
* {
    stroke-geometry: [the_geom];
    stroke:          blue;
    mark-geometry:␣
↪[startPoint(the_geom)], [endPoint(the_geom)];
    mark:␣
↪         symbol(triangle),       symbol(square);
}
```

6.5 YSLD Styling

This module adds support for the YSLD styling language.

YSLD is a YAML based language which closely matches the stucture of SLD, and the internal data model that GeoServer's renderer uses. For details on YSLD syntax see the reference below or the GeoTools documentation.

YSLD is not a part of GeoServer by default, but is available as an optional install.

6.5.1 Installing the GeoServer YSLD extension

1. Download the extension from the nightly GeoServer extension builds.

Warning: Make sure to match the version of the extension to the version of the GeoServer instance!

2. Extract the contents of the archive into the `WEB-INF/lib` directory of the GeoServer installation.

6.5.2 GeoServer Specific Extensions

GeoWebCache Integration

When defining rules in terms of zoom levels, you can use the zoom level from a gridset defined in the integrated GeoWebCache instance.

For instance, if your GWC had a gridset named `CanadaLCCQuad` and you wanted a style rule to apply to levels 0-2 of that gridset you could use the following:

```
grid:
  name: CanadaLCCQuad
rules:
- zoom: [0,2]
  point:
    ...
```

6.5.3 YSLD reference

This section will detail the usage and syntax of the YSLD markup language.

As YSLD is heavily modeled on YAML, it may be useful to refer to the YAML specification for basic syntax.

Structure

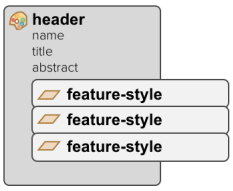
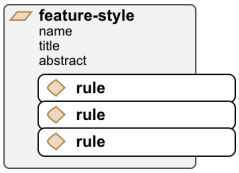
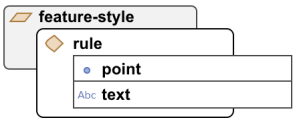
Here is a simple example of a YSLD style containing a single rule inside a single feature style:

```
name: style_example
title: An example of YSLD styling
abstract: Used in the User Manual of GeoServer
feature-styles:
- rules:
  - name: all
    title: Every feature will be styled this way
    symbolizers:
    - polygon:
        fill-color: '#808080'
        fill-opacity: 0.5
        stroke-color: '#000000'
        stroke-opacity: 0.75
```

This would style every polygon feature in a given layer with the given RGB color codes (a medium gray for a fill and black for the outline), with the given opacities for both fill and stroke being given in decimals indicating percentage (so 0.5 is 50% opaque).

Note: For more details on syntax, please see the section on *symbolizers*.

The structure of a typical YSLD file is as follows:

Structure	Description
`Variable definitions`	For common style settings
`Scale grid / zoom levels`	Used to define style based on tile set zoom level
`style header`	Document name, title, and abstract followed by feature-styles 
feature style	Independent block that can contain one or many rules. 
rule	Directive that can contain one or many *symbolizers*. 
filter	A rule "includes" all features unless made to be selective by the use of a *filter*. 
symbolizers	basic unit of a style containing the actual visualization instructions for individual features.

The structure YSLD files is outlined using indentation.

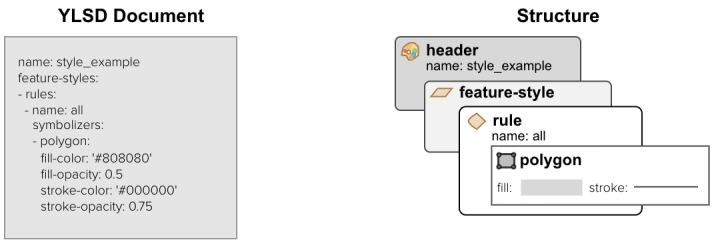


Fig. 6.164: Structure of YSLD style_example

Property syntax

Individual statements (or directives) in a YSLD styling document are designed as key-value, or property-value pairs of the following

form:

```
<property>: <value>
```

The `<property>` is a string denoting the property name, while the `<value>` can be one of a number of different types depending on context. These different types require slightly different markup, as shown in the following table:

Type	Syntax	Example	Notes
Integer	Value only	`12`	Quotes allowed as well
Float	Value only	`0.75`	Quotes allowed as well
Text	Quotes	`Title`	Spaces, colons, and other special characters are allowed. If value is amiguous, use single quotes.
Color	• '# + six digits' (hex) • rgb(r,g,b) (decimal) • Text (named colors)	• `'#FF00FF'` • `rgb(255, 0,255)` • `fuchsia`	Used when specifying RGB colors. For hex, use `'#RRGGBB'` with each two character pair having a value from `00` to `FF`. For decimal, use `rgb(rrr, ggg,bbb)` with each ordinate having a value from `0` to `255`. Quotes are not required when using named colors.
Tuple	Brackets	`[0,15000]`	Use two single quotes to denote blank entries in the tuple (for example: `['#FFFFFF',0,0,'']`).
Filter or other expression	${<expression>}	`${type = road}`	If attribute name is ambiguous, encase in brackets (for example: `${[type] = road}`). If value is ambiguous, use single quotes (`${type = 'road'}`).

Expressions

Throughout the reference guide, there are references to values that are denoted by `<expression>`. An **expression** is a flexible term meaning that the value can be one of the following kinds of objects:

- Literal (scalar or string)
- Attribute name
- *Function*

  If using a function, it must evaluate to match the type expected by the property.

Mappings and lists

Note: The following discussion is taken from basic YAML syntax. Please refer to the YAML specification if necessary.

There are three types of objects in a YSLD document:

1. **Scalar**, a simple value

2. **Mapping**, a collection of key-value (property-value) pairs
3. **List**, any collection of objects. A list can contain mappings, scalars, and even other lists.

   Lists require dashes for every entry, while mappings do not.

   For example, a *symbolizer* block is a list, so every entry requires its own dash:

```
- symbolizer:
  - polygon:
      ...
  - text:
      ...
```

The `point:` and `text:` objects (the individual symbolizers themselves) are mappings, and as such, the contents do not require dashes, only indents:

```
- polygon:
    stroke-color: '#808080'
    fill-color: '#FF0000'
```

The dash next to `polygon` means that the item itself is contained in a list, not that it contains a list. And **the placement of the dash is at the same level of indentation as the list title.**

It is sometimes not obvious whether an object should be a list (and use dashes) or a mapping (and not use dashes), so please refer to this table if unsure:

Object	Type
Feature style	List
Rule	List
Symbolizer	List
Individual symbolizers (contents)	Mapping
Transform	Mapping
Color table (for raster symbolizers)	List

Indentation

Indentation is very important in YSLD. All directives must be indented to its proper place to ensure proper hierarchy. **Improper indentation will cause a style to be rendered incorrectly, or not at all.**

For example, the polygon symbolizer, since it is a mapping, contains certain parameters inside it, such as the color of the fill and stroke. These must be indented such that they are "inside" the polygon block.

In this example, the following markup is **correct**:

```
- polygon:
    fill-color: '#808080'
```

```
    fill-opacity: 0.5
    stroke-color: black
    stroke-opacity: 0.75
```

The parameters inside the polygon (symbolizer) are indented, meaning that they are referencing the symbolizer and are not "outside it."

Compare to the following **incorrect** markup:

```
- polygon:
  fill-color: '#808080'
  fill-opacity: 0.5
  stroke-color: black
  stroke-opacity: 0.75
```

The parameters that are relevant to the polygon block here need to be contained inside that block. Without the parameters being indented, they are at the same "level" as the polygon block, and so will not be interpreted correctly.

Note: For more details on symbolizer syntax, please see the section on *symbolizers*.

Wrapped lines

Long lines can be wrapped by indenting each subsequent line in the text block. New line characters will be converted to spaces, so each line should not end with a space.

So in a situation with a long value:

```
- name: shortname
  title: Longer name
  abstract: This is␣
↪a really long abstract that in no way is ever likely␣
↪to fit on a single line on most people's displays.
```

This can be altered to look like:

```
- name: shortname
  title: Longer name
  abstract:␣
↪This is a really long abstract that in no way
   ␣
↪       is ever likely to fit on a single line on most
           people's displays.
```

In both cases, the value for `abstract` is unchanged.

Wrapped lines can be done between properties and values as well. So this single line:

```
stroke-width: ${roadwidth / 500}
```

Can be altered to look like:

```
stroke-width:
  ${roadwidth / 500}
```

The only constraint with using wrapped lines is that the subsequent lines need to be indented.

Comments

Comments are allowed in YSLD, both for descriptive reasons and to remove certain styling directives without deleting them outright. Comments are indicated by a `#` as the first non-whitespace character in a line. For example:

```
# This is a line symbolizer
- line:
    stroke-color: '#000000'
    stroke-width: 2
#   stroke-width: 3
```

The above would display the lines with width of `2`; the line showing a width of `3` is commented out.

Comment blocks do not exist, so each line of a comment will need to be indicated as such:

```
- line:
    stroke-color: '#000000'
    stroke-width: 2
#- line:
#    stroke-color: '#FF0000'
#    stroke-width: 3
```

Note: Comments are not preserved when converting to SLD.

Feature Styles

In YSLD, a Feature Style is a block of styling *Rules*. The Feature Style is applied to a single feature type and drawn in an off-screen buffer.

Fig. 6.165: The feature style element

The purpose of a Feature Style is to specify drawing order. The buffer for the first Feature Style will be drawn first, while buffer for the second Feature Style will be processed after that, etc. When

drawing is complete the buffers will composed into the final drawn map.

A Feature Style is a **top-level element** in a YSLD style.

Consider the following hierarchy:

- Feature Style 1
  - Rule 1a
  - Rule 1b
- Feature Style 2
  - Rule 2a
  - Rule 2b
  - Rule 2c

In this case, the rules contained inside Feature Style 1 will be processed and their *symbolizers* drawn first. After Rule 1a and 1b are processed, the renderer will move on to Feature Style 2, where Rule 2a, 2b, and 2c will then be processed and their symbolizers drawn.

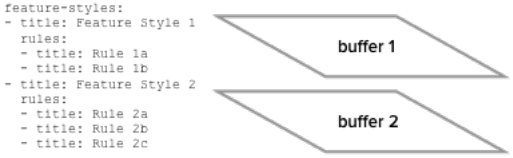


Fig. 6.166: Feature style order

Drawing order

The order of feature styles is significant, and also the order of rules inside feature styles is significant.

Rules inside a feature style are all applied to each feature at once. After all of the rules in a feature style have been applied to each feature, the next feature style will start again, applying rules to each feature.

The off-screen buffer for each feature style is merged together during composition. These buffers are merged in the order defined by the feature styles. In this way, **using multiple feature styles is a way of specifying z-order**.

Consider the same hierarchy as above. Given a layer that contains three features, the rules will be applied as follows:

Feature style 1 will draw an off-screen buffer:

1. Rule 1a is applied to the first feature, followed by rule 1b
2. Rule 1a is applied to the second feature, followed by rule 1b

3. Rule 1a is applied to the third feature, followed by rule 1b

Fig. 6.167: Feature style 1 buffer

Feature style 2 will draw an off-screen buffer:

1. Rule 2a is applied to the first feature, followed by rule 2b and then rule 2c
2. Rule 2a is applied to the second feature, followed by rule 2b and then rule 2c
3. Rule 2a is applied to the third feature, followed by rule 2b and then rule 2c

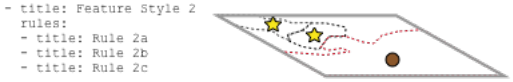


Fig. 6.168: Feature style 2 buffer

This final map is produced by composition:

1. The buffer for feature style 1 is drawn
2. The buffer for feature style 2 is drawn
3. Any labeling is drawn on top

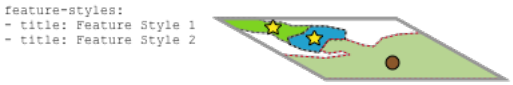


Fig. 6.169: Composition of both feature styles

If you need a rule to apply on top of other rules, use a second feature style. A useful case for this is for lines representing bridges or overpasses. In order to ensure that the bridge lines always display on "top" of other lines (which in a display that includes, they would need to be applied using a second feature style.)

Syntax

The following is the basic syntax of a feature style. Note that the contents of the block are not all expanded here.

```
feature-styles:
- name: <text>
  title: <text>
  abstract: <text>
  transform:
    ...
```

```
rules:
- ...
x-firstMatch: <boolean>
x-composite: <text>
x-composite-base: <boolean>
```

where:

Property	Required?	Description	Default value
`name`	No	Internal reference to the feature style. It is recommended that the value be **lower case** and contain **no spaces**.	Blank
`title`	No	Human-readable name of the feature style. Exposed as a name for the group of rules contained in the feature style.	Blank
`abstract`	No	Longer description of the feature style.	Blank
`transform`	No	*Rendering transformation* information.	N/A
`rules`	Yes	List of styling *rules*.	N/A

The following properties are equivalent to SLD "vendor options".

Property	Required?	Description	Default value
`x-FirstMatch`	No	Stops rule evaluation after the first match. Can make the rendering more efficient by reducing the number of rules that need to be traversed by features, as well as simplyfing the rule filters.	`false`
`x-composite`	No	Allows for both alpha compositing and color blending options between buffers. There are many options; *see below*.	N/A
`x-composite-base`	No	Allows the rendering engine to use that feature-style as a "base", and will compose all subsequent feature-styles and layers on top of it, until another base is found. Once the full set of layers against a base is composed, then the base itself will be composed against the next set of composed layers using its own compositing operator, if present. This is useful to fine-tune the use of `x-composite`, and to make sure that only the desired content is composited/blended and not all of the drawn content.	`false`

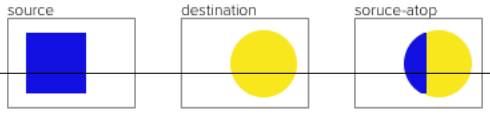
Compositing and blending

By default, multiple feature styles are drawn with one buffer on top of the other. However, using the `x-composite` and `x-composite-base` options, one can customize the way that buffers are displayed.

The following two tables show the possible alpha compositing and color blending values for the `x-composite` option. Note that in the tables below, **source** refers to the buffer that is drawn on top, while **destination** refers to the buffer that the source is drawn on top of.

Alpha compositing

Alpha compositing controls how buffers are merged using the transparent areas of each buffer.

Value	Description
`copy`	Only the source will be present in the output. 
`destination`	Only the destination will be present in the output. 
`source-over`	The source is drawn over the destination, and the destination is visible where the source is transparent. Opposite of `destination-over`. This is the default value for x-composite. 
`destination-over`	The source is drawn below the destination, and is visible only when the destination is transparent. Opposite of `source-over`. 
`source-in`	The source is visible only when overlapping some non-transparent pixel of the destination. This allows the background map to act as a mask for the layer/feature being drawn. Opposite of `destination-in`. 
`destination-in`	The destination is retained only when overlapping some non transparent pixel in the source. This allows the layer/feature to be drawn to act as a mask for the background map. Opposite of `source-in`. 
`source-out`	The source is retained only in areas where the destination is transparent. This acts as a reverse mask when compared to `source-in`. 
`destination-out`	The destination is retained only in areas where the source is transparent. This acts as a reverse mask when compared to `destination-in`. 
`source-atop`	The destination is drawn fully, while the source is drawn only where it intersects the destination. 
`destination-atop`	The source is drawn fully, and the destination is drawn over the source only where it intersects it. 

Color blending

Color blending allows buffers to be mixed during composition.

Value	Description
`multiply`	The source color is multiplied by the destination color and replaces the destination. The resulting color is always at least as dark as either the source or destination color. Multiplying any color with black results in black. Multiplying any color with white preserves the original color. 
`screen`	Multiplies the complements of the source and destination color values, then complements the result. The end result color is always at least as light as either of the two constituent colors. Screening any color with white produces white; screening with black leaves the original color unchanged. 
`overlay`	Multiplies the colors depending on the destination color value. Source colors overlay the destination while preserving highlights and shadows. The backdrop color is not replaced but is mixed with the source color to reflect the lightness or darkness of the backdrop. 
`darken`	Selects the darker of the destination and source colors. The destination is replaced with the source only where the source is darker. 
`lighten`	Selects the lighter of the destination and source colors. The destination is replaced with the source only where the source is lighter. 
`color-dodge`	Brightens the destination color to reflect the source color. Drawing with black produces no changes. 
`color-burn`	Darkens the destination color to reflect the source color. Drawing with white produces no change. 
`hard-light`	Multiplies the colors, depending on the source color value. The effect is similar to shining a harsh spotlight on the destination. 
`soft-light`	Darkens or lightens the colors, depending on the source color value. The effect is similar to a diffused spotlight on the destination. 

Note: For more details about the compositing and blending options, please see the GeoServer User Manual.

Short syntax

When a style has a single feature style, it is possible to omit the syntax for the feature style and start at the first parameter inside.

So the following complete styles are both equivalent:

```
feature-styles:
- rules:
  - name: rule1
    scale: [min,50000]
    symbolizers:
    - line:
        stroke-color: '#000000'
        stroke-width: 2
  - name: rule2
    scale: [50000,max]
    symbolizers:
    - line:
        stroke-color: '#000000'
        stroke-width: 1
```

```
rules:
- name: rule1
  scale: [min,50000]
  symbolizers:
  - line:
      stroke-color: '#000000'
      stroke-width: 2
- name: rule2
  scale: [50000,max]
  symbolizers:
  - line:
      stroke-color: '#000000'
      stroke-width: 1
```

Examples

Road casing

This example shows how a smaller line can be drawn on top of a larger line, creating the effect of lines being drawn with a border or "casing":

```
feature-styles:
- name: outer
  title: Outer line
  rules:
  - name: outer_rule
    symbolizers:
```

```
      - line:
          stroke-color: '#808080'
          stroke-width: 8
- name: inner
  title: Inner line
  rules:
  - name: inner_rule
    symbolizers:
    - line:
        stroke-color: '#44FF88'
        stroke-width: 6
```

To draw the inner lines always on top of the outer lines we need to control the **z-order**. The `outer_rule` is encased in its own feature style and drawn into a distinct "Outer line" buffer. Next the `inner_rule` is encased in its own feature style and drawn into a distinct "Inner line" buffer.

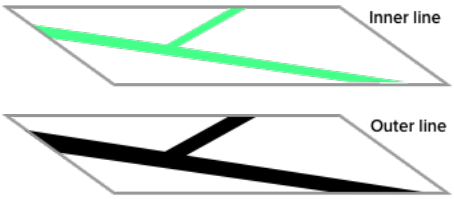


Fig. 6.170: Feature style buffers

During composition these two off-screen buffers are combined into the the final map.

Fig. 6.171: Final map composition

When drawn, the outer line has a width of 8 pixels and the inner line has a width of 6 pixels, so the line "border" is 1 pixel (on each side).

First match

Given a style that has many rules with distinct outcomes, it may be advantageous to employ `x-firstMatch` so as to improve rendering efficiency and simplify those rules.

This first example shows the standard way of creating rules for a dataset. There are villages, towns, and cities (`type = 'village'`, `type = 'town'` or `type = 'city'`) and they have an `industry` which could be either `fishing` or other values.

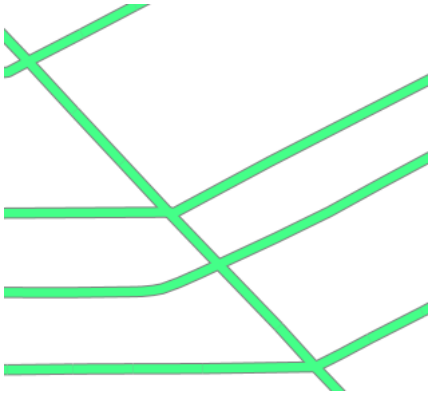


Fig. 6.172: Example showing road casing

Note: In order to simplify this example, the specifics of the point symbolizers have been replaced by *Variables*. In a real-world example, these would need to be defined in the YSLD as well.

```
feature-styles:
- name: without_first_match
  rules:
  - name: fishing_town
    filter: ${type = 'town' AND industry = 'fishing'}
    symbolizers:
    - point:
        <<: *fishingtown
  - name: fishing_city
    filter: ${type = 'city' AND industry = 'fishing'}
    symbolizers:
    - point:
        <<: *fishingcity
  - name: other_towns_cities
    filter:
↪${type IN ('town', 'city') AND industry <> 'fishing'}
    symbolizers:
    - point:
        <<: *othertownscities
  - name: other
    else: true
    symbolizers:
    - point:
        <<: *allotherplaces
```

Using the `x-firstMatch: true` parameter, the style is simplified:

```
feature-styles:
- name: with_first_match
  x-firstMatch: true
  rules:
  - name: fishing_town
    filter: ${type = 'town' AND industry = 'fishing'}
    symbolizers:
    - point:
```

```
      <<: *fishingtown
  - name: fishing_city
    filter: ${type = 'city' AND industry = 'fishing'}
    symbolizers:
    - point:
        <<: *fishingcity
  - name: other_towns_cities
    filter: ${type IN ('town', 'city')}
    symbolizers:
    - point:
        <<: *othertownscities
  - name: other
    else: true
    symbolizers:
    - point:
        <<: *allotherplaces
```

Specifically, the third rule no longer needs the extra `AND industry <> 'fishing'`, because the previous two rules imply that any features remaining by this rule have that condition.

Layer mask

Given two layers (in this case, two three-band rasters), one can mask or "knock out" the other, making visible what's beneath.

Fig. 6.173: Top/source layer

Note: Screenshots show data provided by Natural Earth.

Layer 1 (top/source):

```
feature-styles:
- rules:
  - title: Top/source
    symbolizers:
    - raster:
```

Fig. 6.174: Bottom/destination layer

```
      opacity: 1.0
    x-composite: xor
```

Layer 2 (bottom/destination):

```
feature-styles:
- rules:
  - title: Bottom/destination
    symbolizers:
    - raster:
        opacity: 1.0
```

Fig. 6.175: Layer as mask

Color inversion

Given the same two layers as the previous example, one can display the difference of the colors of layers, which can have the effect of a color "inversion".

Layer 1 (top/source):

```
feature-styles:
- rules:
  - title: Top/source
    symbolizers:
    - raster:
        opacity: 1.0
  x-composite: difference
```

Layer 2 (bottom/destination):

```
feature-styles:
- rules:
  - title: Bottom/destination
    symbolizers:
    - raster:
        opacity: 1.0
```

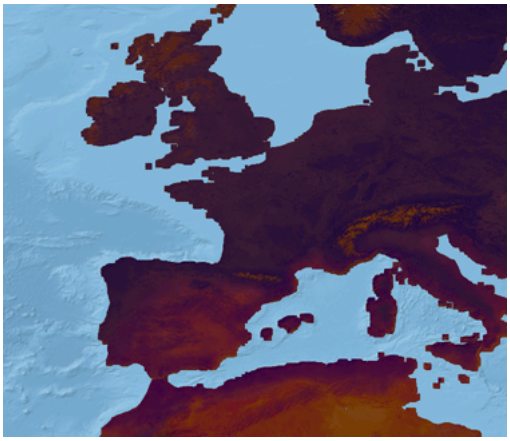


Fig. 6.176: Layer as color inversion

Rules

A rule is a **collection of styling directives**, primarily consisting of *symbolizers* combined with optional conditional statements.

- **If a conditional statement exists** in a rule, then the styling directives will only be carried out **if the conditional returns true**. Otherwise, the rule will be skipped.
- **If no conditional statement exists** in a rule, then the styling directive will **always be carried out**.

The types of conditional statements available to rules are:

- *Filters* for attribute-based rendering
- *Scale* for scale-based rendering

Rules are contained within *feature styles*. There is no limit on the number of rules that can be created, and there is no restriction that all rules must be mutually exclusive (as in, some rules may apply to the same features).

Syntax

The following is the basic syntax of a rule. Note that the contents of the block are not all expanded; see the other sections for the relevant syntax.

```
rules:
- name: <text>
  title: <text>
  filter: <filter>
  else: <boolean>
  scale: [<min>,<max>]
  symbolizers:
  - ...
```

where:

Property	Required?	Description	Default value
`name`	No	Internal reference to the feature style. It is recommended that the value be **lower case** and contain **no spaces**.	Blank
`title`	No	Human-readable description of what the rule accomplishes.	Blank
`filter`	No	*Filter* expression which will need to evaluate to be true for the symbolizer(s) to be applied. Cannot be used with `else`.	Blank (meaning that the rule will apply to all features)
`else`	No	Specifies whether the rule will be an "else" rule. An else rule applies when, after scale and filters are applied, no other rule applies. To make an else rule, set this option to `true`. Cannot be used with `filter`.	`false`
`scale`	No	*Scale* boundaries showing at what scales (related to zoom levels) the rule will be applied.	Visible at all scales
`symbolizers`	Yes	Block containing one or more *symbolizers*. These contain the actual visualization directives. If the filter returns true and the view is within the scale boundaries, these symbolizers will be drawn.	N/A

Short syntax

When a style has a single rule inside a single feature style, it is possible to omit the syntax for both and start at the first parameter inside.

So the following complete styles are equivalent:

```
feature-styles:
- rules:
  - symbolizers:
    - line:
        stroke-color: '#000000'
        stroke-width: 2
```

```
line:
  stroke-color: '#000000'
  stroke-width: 2
```

Examples

Else filter

Using `filter` and `else` together:

```
rules:
- name: small
  title: Small features
  filter: ${type = 'small'}
  symbolizers:
  - ...
- name: large
  title: Large features
  filter: ${type = 'large'}
  symbolizers:
  - ...
- name: else
  title: All other features
  else: true
  symbolizers:
  - ...
```

In the above situation:

- If a feature has a value of "small" in its `type` attribute, it will be styled with the "small" rule.
- If a feature has a value of "large" in its `type` attribute, it will be styled with the "large" rule.
- If a feature has a value of "medium" (or anything else) in its `type` attribute, it will be styled with the "else" rule.

Else with scale

Using `filter`, `else`, and `scale` together:

```
rules:
- name: small_zoomin
  scale: [min,10000]
  title: Small features when zoomed in
  filter: ${type = 'small'}
  symbolizers:
  - ...
- name: small_zoomout
  scale: [10000,max]
  title: Small features when zoomed out
  filter: ${type = 'small'}
  symbolizers:
  - ...
```

```
  - name: else_zoomin
    scale: [min,10000]
    title: All other features when zoomed in
    else: true
    symbolizers:
    - ...
  - name: else_zoomout
    scale: [10000,max]
    title: All other features when zoomed out
    else: true
    symbolizers:
    - ...
```

In the above situation:

- If a feature has a value of "small" in its `type` attribute, and the map is a scale level less than 10,000, it will be styled with the "small_zoomin" rule.
- If a feature has a value of anything else other than "small" in its `type` attribute, and the map is a scale level less than 10,000, it will be styled with the "else_zoomin" rule.
- If a feature has a value of "small" in its `type` attribute, and the map is a scale level greater than 10,000, it will be styled with the "small_zoomout" rule.
- If a feature has a value of anything else other than "small" in its `type` attribute, and the map is a scale level greater than 10,000, it will be styled with the "else_zoomout" rule.

Symbolizers

The basic unit of visualization is the symbolizer. There are five types of symbolizers: **Point**, **Line**, **Polygon**, **Raster**, and **Text**.

Symbolizers are contained inside *rules*. A rule can contain one or many symbolizers.

Note: The most common use case for multiple symbolizers is a geometry (point/line/polygon) symbolizer to draw the features plus a text symbolizer for labeling these features.

Fig. 6.177: Use of multiple symbolizers

Drawing order

The order of symbolizers significant, and also the order of your data.

For each feature the rules are evaluated resulting in a list of symbolizers that will be used to draw that feature. The symbolizers are drawn in the order provided.

Consider the following two symbolizers:

```
symbolizers:
- point:
    symbols:
    - mark:
        shape: square
        fill-color: '#FFCC00'
- point:
    symbols:
    - mark:
        shape: triangle
        fill-color: '#FF3300'
```

When drawing three points these symbolizers will be applied in order on each feature:

1. Feature 1 is drawn as a square, followed by a triangle:

Fig. 6.178: Feature 1 buffer rendering

2. Feature 2 is drawn as a square, followed by a triangle. Notice the slight overlap with Feature 1:

Fig. 6.179: Feature 2 buffer rendering

3. Feature 3 is drawn as a square, followed by a triangle:

Fig. 6.180: Feature 3 buffer rendering

Note: In the final image, Feature 1 and Feature 2 have a slight overlap. This overlap is determined by data order which we have no control over. If you need to control the overlap review the *Feature Styles* section on managing "z-order".

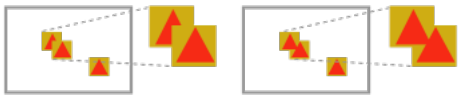


Fig. 6.181: Feature style controlling z-order

Matching symbolizers and geometries

It is common to match the symbolizer with the type of geometries contained in the layer, but this is not required. The following table illustrates what will happen when a geometry symbolizer is matched up with another type of geometry.

	Points	Lines	Polygon	Raster
Point Symbolizer	**Points**	Midpoint of the lines	Centroid of the polygons	Centroid of the raster
Line Symbolizer	n/a	**Lines**	Outline (stroke) of the polygons	Outline (stroke) of the raster
Polygon Symbolizer	n/a	Will "close" the line and style as a polygon	**Polygons**	Will "outline" the raster and style as a polygon
Raster Symbolizer	n/a	n/a	n/a	Transform raster values to color channels for display
Text Symbolizer	Label at point location	Label at midpoint of lines	Label at centroid of polygons	Label at centroid of raster outline

Syntax

The following is the basic syntax common to all symbolizers. Note that the contents of the block are not all expanded here and that each kind of symbolizer provides additional syntax.

```
geometry: <cql>
uom: <text>
..
x-composite: <text>
x-composite-base: <boolean>
```

where:

Property	Required?	Description	Default value
`geometry`	No	Specifies which attribute to use as the geometry.	First geometry attribute found (usually named `geom` or `the_geom`)
`uom`	No	Unit of measure used for width calculations	pixel

The following properties are equivalent to SLD "vendor options".

Property	Required?	Description	Default value
`x-composite`	No	Allows for both alpha compositing and color blending options between symbolizers.	N/A
`x-composite-base`	No	Allows the rendering engine to use the symbolizer mapping to define a "base" buffer for subsequent compositing and blending using `x-composite`. See the section on *Feature Styles* for more details.	`false`

See the following pages for details:

Line symbolizer

The line symbolizer is used to style linear (1-dimensional) features. It is in some ways the simplest of the symbolizers because it only contains facilities for the stroke (outline) of a feature.

Syntax

The full syntax of a line symbolizer is:

```
symbolizers:
- line:
    stroke-color: <color>
    stroke-width: <expression>
    stroke-opacity: <expression>
    stroke-linejoin: <expression>
    stroke-linecap: <expression>
    stroke-dasharray: <float list>
    stroke-dashoffset: <expression>
    stroke-graphic:
      <graphic_options>
    stroke-graphic-fill:
      <graphic_options>
    offset: <expression>
    geometry: <expression>
    uom: <text>
    x-labelObstacle: <boolean>
    x-composite-base: <boolean>
    x-composite: <text>
```

where:

Property	Required?	Description	Default value
`stroke-color`	No	Color of line features.	`'#000000'` (black)
`stroke-width`	No	Width of line features, measured in pixels.	`1`
`stroke-opacity`	No	Opacity of line features. Valid values are a decimal value between `0` (completely transparent) and `1` (completely opaque).	`1`
`stroke-linejoin`	No	How line segments are joined together. Options are `mitre` (sharp corner), `round` (round corner), and `bevel` (diagonal corner).	`mitre`
`stroke-linecap`	No	How line features are rendered at their ends. Options are `butt` (sharp square edge), `round` (rounded edge), and `square` (slightly elongated square edge).	`butt`
`stroke-dasharray`	No	A numeric list signifying length of lines and gaps, creating a dashed effect. Units are pixels, so `"2 3"` would be a repeating pattern of 2 pixels of drawn line followed by 3 pixels of blank space. If only one number is supplied, this will mean equal amounts of line and gap.	No dash
`stroke-dashoffset`	No	Number of pixels into the dasharray to offset the drawing of the dash, used to shift the location of the lines and gaps in a dash.	`0`
`stroke-graphic`	No	A design or pattern to be used along the stroke. Output will always be a linear repeating pattern, and as such is not tied to the value of `stroke-width`. Can either be a mark consisting of a common shape or a URL that points to a graphic. The `<graphic_options>` should consist of a mapping containing `symbols:` followed by an `external:` or `mark:`, with appropriate parameters as detailed in the *Point symbolizer* section. Cannot be used with `stroke-graphic-fill`.	N/A
`stroke-graphic-fill`	No	A design or pattern to be used for the fill of the stroke. The area that is to be filled is tied directly to the value of `stroke-width`. Can either be a mark consisting of a common shape or a URL that points to a graphic. The `<graphic_options>` should consist of a mapping containing `symbols:` followed by an `external:` or `mark:`, with appropriate parameters as detailed in the *Point symbolizer* section. Cannot be used with `stroke-graphic`.	N/A

Property	Required?	Description	Default value
`offset`	No	Value in pixels for moving the drawn line relative to the location of the feature.	`0`

Property	Required?	Description	Default value
`geometry`	No	Specifies which attribute to use as the geometry.	First geometry attribute found (usually named `geom` or `the_geom`)
`uom`	No	Unit of measure used for width calculations	pixel

The following properties are equivalent to SLD "vendor options".

Property	Required?	Description	Default value
`x-labelObstacle`	No	Marks the symbolizer as an obstacle such that labels drawn via a *text symbolizer* will not be drawn over top of these features. Options are `true` or `false`. Note that the bounding boxes of features are used when calculating obstacles, so unintended effects may occur when marking a line or polygon symbolizer as an obstacle.	`false`

Property	Required?	Description	Default value
`x-composite`	No	Allows for both alpha compositing and color blending options between symbolizers.	N/A
`x-composite-base`	No	Allows the rendering engine to use the symbolizer mapping to define a "base" buffer for subsequent compositing and blending using `x-composite`. See the section on *Feature Styles* for more details.	`false`

Examples

Basic line with styled ends

The `linejoin` and `linecap` properties can be used to style the joins and ends of any stroke. This example draws lines with partially transparent black lines with rounded ends and sharp (mitred) corners:

```
feature-styles:
- rules:
  - symbolizers:
    - line:
        stroke-color: '#000000'
        stroke-width: 8
        stroke-opacity: 0.5
        stroke-linejoin: mitre
        stroke-linecap: round
```

Railroad pattern

Many maps use a hatched pattern to represent railroads. This can be accomplished by using two line symbolizers, one solid and one

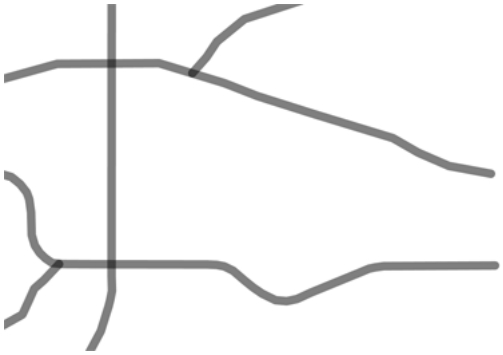


Fig. 6.182: Basic line with styled ends

dashed. Specifically, the `stroke-dasharray` property is used to create a dashed line of length 1 every 24 pixels:

```
name: railroad
feature-styles:
- name: name
  rules:
  - symbolizers:
    - line:
        stroke-color: '#000000'
        stroke-width: 1
    - line:
        stroke-color: '#000000'
        stroke-width: 12
        stroke-dasharray: '1 24'
```

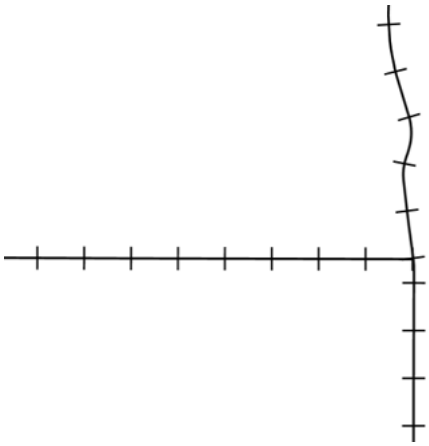


Fig. 6.183: Railroad pattern

Specifying sizes in units

The units for `stroke-width`, `size`, and other similar attributes default to pixels, meaning that graphics remain a constant size at different zoom levels. Alternately, units (feet or meters) can be specified for values, so graphics will scale as you zoom in or out. This

example draws roads with a fixed width of 8 meters:

```
feature-styles:
- rules:
  - symbolizers:
    - line:
        stroke-color: '#000000'
        stroke-width: '8 m'
```

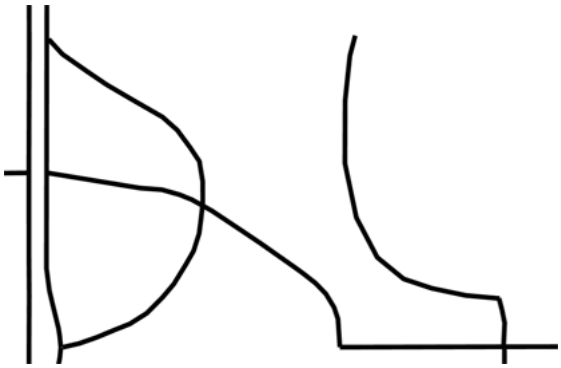


Fig. 6.184: Line width measured in meters (zoomed out)

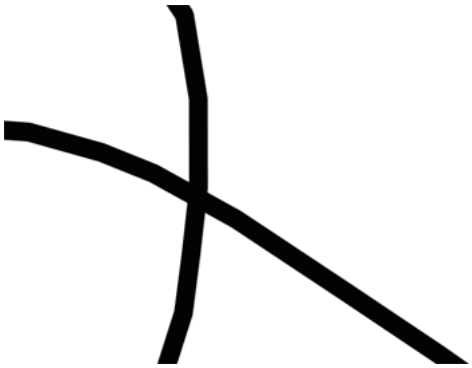


Fig. 6.185: Line width measured in meters (zoomed in)

The default unit of measure for the symbolizer is defined using `uom`. This example uses a default of meters to supply distances for `stroke-width` and `stroke-dasharray` using meters.

```
line:
  uom: metre
  stroke-color: '#000000'
  stroke-width: '8'
  stroke-dasharray: '20 3'
```

Polygon symbolizer

The polygon symbolizer styles polygon (2-dimensional) features. It contains facilities for the stroke (outline) of a feature as well as the fill (inside) of a feature.

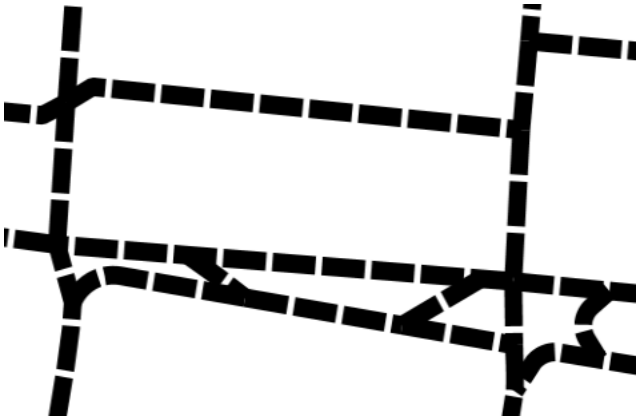


Fig. 6.186: Line width and spacing in meters

Syntax

The full syntax of a polygon symbolizer is:

```
symbolizers:
- polygon:
    fill-color: <color>
    fill-opacity: <expression>
    fill-graphic:
      <graphic_options>
    stroke-color: <color>
    stroke-width: <expression>
    stroke-opacity: <expression>
    stroke-linejoin: <expression>
    stroke-linecap: <expression>
    stroke-dasharray: <float list>
    stroke-dashoffset: <expression>
    stroke-graphic:
      <graphic_options>
    stroke-graphic-fill:
      <graphic_options>
    offset: <expression>
    displacement: <expression>
    geometry: <expression>
    uom: <text>
    x-labelObstacle: <boolean>
    x-composite-base: <boolean>
    x-composite: <text>
```

where:

Property	Required?	Description	Default value
`stroke-color`	No	Color of line features.	`'#000000'` (black)
`stroke-width`	No	Width of line features, measured in pixels.	`1`
`stroke-opacity`	No	Opacity of line features. Valid values are a decimal value between `0` (completely transparent) and `1` (completely opaque).	`1`
`stroke-linejoin`	No	How line segments are joined together. Options are `mitre` (sharp corner), `round` (round corner), and `bevel` (diagonal corner).	`mitre`
`stroke-linecap`	No	How line features are rendered at their ends. Options are `butt` (sharp square edge), `round` (rounded edge), and `square` (slightly elongated square edge).	`butt`
`stroke-dasharray`	No	A numeric list signifying length of lines and gaps, creating a dashed effect. Units are pixels, so `"2 3"` would be a repeating pattern of 2 pixels of drawn line followed by 3 pixels of blank space. If only one number is supplied, this will mean equal amounts of line and gap.	No dash
`stroke-dashoffset`	No	Number of pixels into the dasharray to offset the drawing of the dash, used to shift the location of the lines and gaps in a dash.	`0`
`stroke-graphic`	No	A design or pattern to be used along the stroke. Output will always be a linear repeating pattern, and as such is not tied to the value of `stroke-width`. Can either be a mark consisting of a common shape or a URL that points to a graphic. The `<graphic_options>` should consist of a mapping containing `symbols:` followed by an `external:` or `mark:`, with appropriate parameters as detailed in the *Point symbolizer* section. Cannot be used with `stroke-graphic-fill`.	N/A
`stroke-graphic-fill`	No	A design or pattern to be used for the fill of the stroke. The area that is to be filled is tied directly to the value of `stroke-width`. Can either be a mark consisting of a common shape or a URL that points to a graphic. The `<graphic_options>` should consist of a mapping containing `symbols:` followed by an `external:` or `mark:`, with appropriate parameters as detailed in the *Point symbolizer* section. Cannot be used with `stroke-graphic`.	N/A

Property	Required?	Description	Default value
`fill-color`	No	Color of inside of features.	`'#808080'` (gray)
`fill-opacity`	No	Opacity of the fill. Valid values are a decimal value between `0` (completely transparent) and `1` (completely opaque).	`1`
`fill-graphic`	No	A design or pattern to be used for the fill of the feature. Can either be a mark consisting of a common shape or a URL that points to a graphic. The `<graphic_options>` should consist of a mapping containing `symbols:` followed by an `external:` or `mark:`, with appropriate parameters as detailed in the *Point symbolizer* section.	None

The use of `fill-graphic` allows for the following extra options:

Property	Required?	Description	Default value
`x-graphic-margin`	No	Used to specify margins (in pixels) around the graphic used in the fill. Possible values are a list of four (`top, right, bottom, left`), a list of three (`top, right and left, bottom`), a list of two (`top and bottom, right and left`), or a single value.	N/A
`x-random`	No	Activates random distribution of symbols. Possible values are `free` or `grid`. `free` generates a completely random distribution, and `grid` will generate a regular grid of positions, and only randomize the position of the symbol around the cell centers, providing a more even distribution.	N/A
`x-random-tile-size`	No	When used with `x-random`, determines the size of the grid (in pixels) that will contain the randomly distributed symbols.	`256`
`x-random-rotation`	No	When used with `x-random`, activates random symbol rotation. Possible values are `none` or `free`.	`none`
`x-random-symbol-count`	No	When used tih `x-random`, determines the number of symbols drawn. Increasing this number will generate a more dense distribution of symbols	`16`
`x-random-seed`	No	Determines the "seed" used to generate the random distribution. Changing this value will result in a different symbol distribution.	`0`

Property	Required?	Description	Default value
`offset`	No	Value in pixels for moving the drawn line relative to the location of the feature.	`0`
`displacement`	No	Specifies a distance to which to move the symbol relative to the feature. Value is an `[x,y]` tuple with values expressed in pixels, so [10,5] will displace the symbol 10 pixels to the right, and 5 pixels down.	`[0,0]`

Property	Required?	Description	Default value
`geometry`	No	Specifies which attribute to use as the geometry.	First geometry attribute found (usually named `geom` or `the_geom`)
`uom`	No	Unit of measure used for width calculations	pixel

The following properties are equivalent to SLD "vendor options".

Property	Required?	Description	Default value
`x-labelObstacle`	No	Marks the symbolizer as an obstacle such that labels drawn via a *text symbolizer* will not be drawn over top of these features. Options are `true` or `false`. Note that the bounding boxes of features are used when calculating obstacles, so unintended effects may occur when marking a line or polygon symbolizer as an obstacle.	`false`

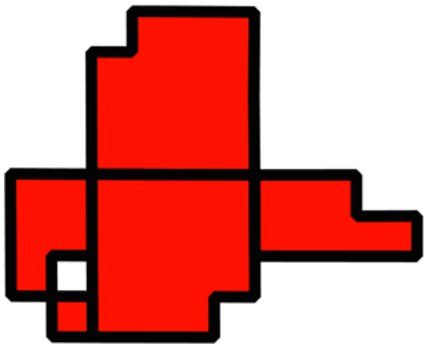
Property	Required?	Description	Default value
`x-composite`	No	Allows for both alpha compositing and color blending options between symbolizers.	N/A
`x-composite-base`	No	Allows the rendering engine to use the symbolizer mapping to define a "base" buffer for subsequent compositing and blending using `x-composite`. See the section on *Feature Styles* for more details.	`false`

Examples

Basic polygon

Polygon symbolizers have both a stroke and a fill, similar to marks for point symbolizers. The following example draws a polygon symbolizer with a red fill and black stroke with beveled line joins for the stroke:

```
feature-styles:
- name: name
  rules:
  - title: fill-graphic
    symbolizers:
    - polygon:
        fill-color: '#FF0000'
        fill-opacity: 0.9
        stroke-color: '#000000'
        stroke-width: 8
        stroke-opacity: 1
        stroke-linejoin: bevel
```



Fill with graphic

The `fill-graphic` property is used to fill a geometry with a repeating graphic. This can be a mark or an external image. The `x-graphic-margin` option can be used to specify top, right, bottom, and left margins around the graphic used in the fill. This example uses two sets of repeating squares with different offset values to draw a checkerboard pattern:

```
name: checkers
feature-styles:
- name: name
  rules:
  - title: fill-graphic
    symbolizers:
    - polygon:
        stroke-width: 1
        fill-graphic:
          symbols:
          - mark:
              shape: square
              fill-color: '#000000'
          size: 8
        x-graphic-margin: 16 16 0 0
    - polygon:
        stroke-width: 1
        fill-graphic:
          symbols:
          - mark:
              shape: square
              fill-color: '#000000'
          size: 8
        x-graphic-margin: 0 0 16 16
```

Randomized graphic fill

Normally, the graphic used for the `fill-graphic` property is tiled. Alternatively, one can scatter this image randomly across the fill area using the `x-random` option and associated other options. This

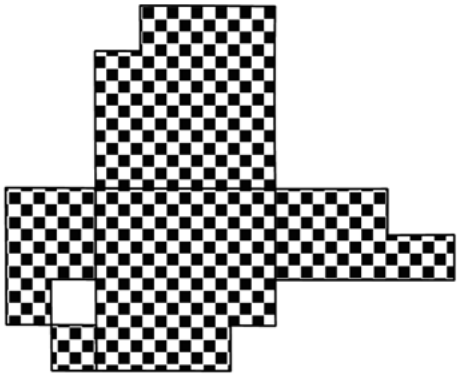


Fig. 6.187: Checkered fill

could be used to create a speckled pattern, as in the following example:

```
name: speckles
feature-styles:
- name: name
  rules:
  - title: fill-graphic
    symbolizers:
    - polygon:
        stroke-width: 1
        fill-graphic:
          symbols:
            - mark:
                shape: circle
                fill-color: '#000000'
          size: 3
          x-random: grid
          x-random-seed: 2
          x-random-tile-size: 1000
          x-random-rotation: free
          x-random-symbol-count: 1000
```

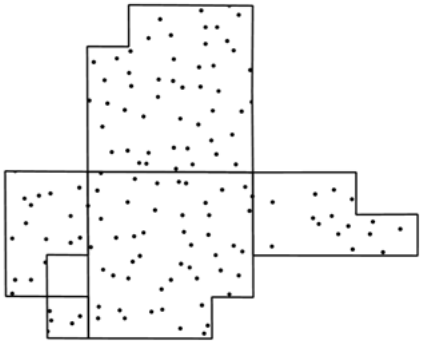


Fig. 6.188: Randomized graphic fill

Point symbolizer

The point symbolizer is used to style point features or centroids of non-point features.

Syntax

The full syntax of a point symbolizer is:

```
symbolizers:
- point:
    symbols:
    - external:
        url: <text>
        format: <text>
    - mark:
        shape: <shape>
        fill-color: <color>
        fill-opacity: <expression>
        fill-graphic:
          <graphic_options>
        stroke-color: <color>
        stroke-width: <expression>
        stroke-opacity: <expression>
        stroke-linejoin: <expression>
        stroke-linecap: <expression>
        stroke-dasharray: <float list>
        stroke-dashoffset: <expression>
        stroke-graphic:
          <graphic_options>
        stroke-graphic-fill:
          <graphic_options>
    size: <expression>
    anchor: <tuple>
    displacement: <tuple>
    opacity: <expression>
    rotation: <expression>
    geometry: <expression>
    uom: <text>
    x-labelObstacle: <boolean>
    x-composite-base: <boolean>
    x-composite: <text>
```

where:

Property	Required?	Description	Default value
`stroke-color`	No	Color of line features.	`'#000000'` (black)
`stroke-width`	No	Width of line features, measured in pixels.	`1`
`stroke-opacity`	No	Opacity of line features. Valid values are a decimal value between `0` (completely transparent) and `1` (completely opaque).	`1`
`stroke-linejoin`	No	How line segments are joined together. Options are `mitre` (sharp corner), `round` (round corner), and `bevel` (diagonal corner).	`mitre`
`stroke-linecap`	No	How line features are rendered at their ends. Options are `butt` (sharp square edge), `round` (rounded edge), and `square` (slightly elongated square edge).	`butt`
`stroke-dasharray`	No	A numeric list signifying length of lines and gaps, creating a dashed effect. Units are pixels, so `"2 3"` would be a repeating pattern of 2 pixels of drawn line followed by 3 pixels of blank space. If only one number is supplied, this will mean equal amounts of line and gap.	No dash
`stroke-dashoffset`	No	Number of pixels into the dasharray to offset the drawing of the dash, used to shift the location of the lines and gaps in a dash.	`0`
`stroke-graphic`	No	A design or pattern to be used along the stroke. Output will always be a linear repeating pattern, and as such is not tied to the value of `stroke-width`. Can either be a mark consisting of a common shape or a URL that points to a graphic. The `<graphic_options>` should consist of a mapping containing `symbols:` followed by an `external:` or `mark:`, with appropriate parameters as detailed in the *Point symbolizer* section. Cannot be used with `stroke-graphic-fill`.	N/A
`stroke-graphic-fill`	No	A design or pattern to be used for the fill of the stroke. The area that is to be filled is tied directly to the value of `stroke-width`. Can either be a mark consisting of a common shape or a URL that points to a graphic. The `<graphic_options>` should consist of a mapping containing `symbols:` followed by an `external:` or `mark:`, with appropriate parameters as detailed in the *Point symbolizer* section. Cannot be used with `stroke-graphic`.	N/A

Property	Required?	Description	Default value
`fill-color`	No	Color of inside of features.	`'#808080'` (gray)
`fill-opacity`	No	Opacity of the fill. Valid values are a decimal value between `0` (completely transparent) and `1` (completely opaque).	`1`
`fill-graphic`	No	A design or pattern to be used for the fill of the feature. Can either be a mark consisting of a common shape or a URL that points to a graphic. The `<graphic_options>` should consist of a mapping containing `symbols:` followed by an `external:` or `mark:`, with appropriate parameters as detailed in the *Point symbolizer* section.	None

The use of `fill-graphic` allows for the following extra options:

Property	Required?	Description	Default value
`x-graphic-margin`	No	Used to specify margins (in pixels) around the graphic used in the fill. Possible values are a list of four (`top, right, bottom, left`), a list of three (`top, right and left, bottom`), a list of two (`top and bottom, right and left`), or a single value.	N/A
`x-random`	No	Activates random distribution of symbols. Possible values are `free` or `grid`. `free` generates a completely random distribution, and `grid` will generate a regular grid of positions, and only randomize the position of the symbol around the cell centers, providing a more even distribution.	N/A
`x-random-tile-size`	No	When used with `x-random`, determines the size of the grid (in pixels) that will contain the randomly distributed symbols.	`256`
`x-random-rotation`	No	When used with `x-random`, activates random symbol rotation. Possible values are `none` or `free`.	`none`
`x-random-symbol-count`	No	When used tih `x-random`, determines the number of symbols drawn. Increasing this number will generate a more dense distribution of symbols	`16`
`x-random-seed`	No	Determines the "seed" used to generate the random distribution. Changing this value will result in a different symbol distribution.	`0`

Property	Required?	Description	Default value
`external`	No	Specifies an image to use to style the point.	N/A
`url`	Yes	Location of the image. Can either be an actual URL or a file path (relative to where the style file is saved in the GeoServer data directory). Should be enclosed in single quotes.	N/A
`format`	Yes	Format of the image. Must be a valid MIME type (such as `image/png` for PNG, `image/jpeg` for JPG, `image/svg+xml` for SVG)	N/A
`mark`	No	Specifies a regular shape to use to style the point.	N/A
`shape`	No	Shape of the mark. Options are `square`, `circle`, `triangle`, `cross`, `x`, and `star`.	`square`
`size`	No	Size of the mark in pixels. If the aspect ratio of the mark is not 1:1 (square), will apply to the *height* of the graphic only, with the width scaled proportionally.	16
`anchor`	No	Specify the center of the symbol relative to the feature location. Value is an `[x,y]` tuple with decimal values from 0-1, with `[0,0]` meaning that the symbol is anchored to the top left, and `[1,1]` meaning anchored to bottom right.	`[0.5,0.5]`
`displacement`	No	Specifies a distance to which to move the symbol relative to the feature. Value is an `[x,y]` tuple with values expressed in pixels, so [10,5] will displace the symbol 10 pixels to the right and 5 pixels down.	`[0,0]`
`opacity`	No	Specifies the level of transparency. Value of `0` means entirely transparent, while `1` means entirely opaque. Only affects graphics referenced by the `external` parameter; the opacity of `mark` symbols is controled by the `fill-opacity` and `stroke-opacity` of the mark.	`1`
`rotation`	No	Value (in degrees) or rotation of the mark. Larger values increase counter-clockwise rotation. A value of `180` will make the mark upside-down.	`0`

Property	Required?	Description	Default value
`geometry`	No	Specifies which attribute to use as the geometry.	First geometry attribute found (usually named `geom` or `the_geom`)
`uom`	No	Unit of measure used for width calculations	pixel

The following properties are equivalent to SLD "vendor options".

Property	Required?	Description	Default value
`x-labelObstacle`	No	Marks the symbolizer as an obstacle such that labels drawn via a *text symbolizer* will not be drawn over top of these features. Options are `true` or `false`. Note that the bounding boxes of features are used when calculating obstacles, so unintended effects may occur when marking a line or polygon symbolizer as an obstacle.	`false`

Property	Required?	Description	Default value
`x-composite`	No	Allows for both alpha compositing and color blending options between symbolizers.	N/A
`x-composite-base`	No	Allows the rendering engine to use the symbolizer mapping to define a "base" buffer for subsequent compositing and blending using `x-composite`. See the section on *Feature Styles* for more details.	`false`

Examples

Basic point

A point symbolizer draws a point at the center of any geometry. It is defined by an external image or a symbol, either of which can be sized and rotated. A mark is a pre-defined symbol that can be drawn at the location of a point. Similar to polygons, marks have both a fill and a stroke. This example shows a point symbolizer that draws semi-transparent red diamonds with black outlines:

```
feature-styles:
- name: name
  rules:
  - title: red point
    symbolizers:
    - point:
        symbols:
        - mark:
            shape: square
            fill-color: '#FF0000'
            fill-opacity: 0.75
            stroke-color: '#000000'
            stroke-width: 1.5
            stroke-opacity: 1
        size: 20
        rotation: 45
```

Point as image

Sometimes it may be useful to use an image to represent certain points. This can be accomplished using the `external` symbol property, which requires a `url` and a `format`. The `url` should be enclosed in single quotes. The `format` property is a MIME type im-

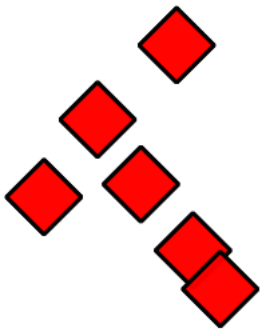


Fig. 6.189: Basic point

age. This example shows a point symbolizer that draws an image centered on each point:

```
name: point
feature-styles:
- name: name
  rules:
  - symbolizers:
    - point:
        symbols:
        - external:
            url: 'geoserver.png'
            format: image/png
        size: 16
```

Fig. 6.190: Point as image

Point composition

Using more than one point symbolizer allows the composition of more complex symbology. This example shows two symbolizers along with the `x-composite` parameter in order to *subtract* a shape from a square mark, allowing the background to show through.

```
symbolizers:
- point:
    symbols:
    - mark:
        shape: square
        fill-color: '#222222'
    size: 40
- point:
    symbols:
    - external:
        url: 'stamp.png'
        format: image/png
    x-composite: xor
    size: 40
```

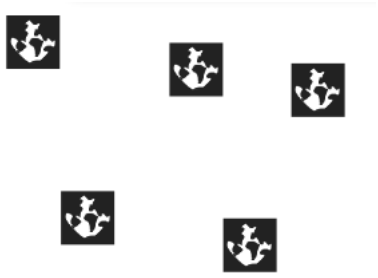


Fig. 6.191: Point composition

Points as arrow heads

Sometimes it is useful to generate a point using a CQL expression. The following example generates a point at the end of each line in the shape of an arrow, rotated such that it matches the orientation of the line.

```
name: arrow
symbolizers:
- line:
   stroke-color: '#808080'
   stroke-width: 3
- point:
    geometry: ${endPoint(the_geom)}
    symbols:
    - mark:
        shape: shape://oarrow
        fill-color: '#808080'
    size: 30
    rotation: ${endAngle(the_geom)}
```

Raster symbolizer

The raster symbolizer styles raster (coverage) layers. A raster is an array of information with each cell in the array containing one or more values, stored as "bands".

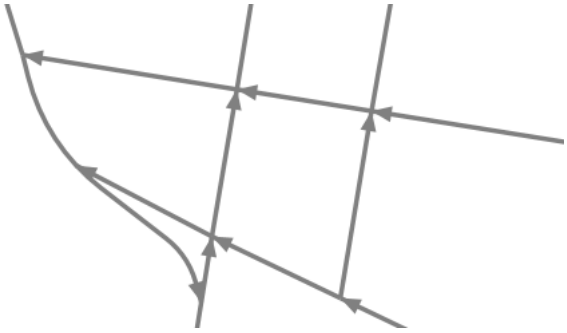


Fig. 6.192: Point as arrow head

The full syntax of a raster symbolizer is:

```
symbolizers:
- raster:
    opacity: <expression>
    channels:
      gray:
        <channel_options>
      red:
        <channel_options>
      green:
        <channel_options>
      blue:
        <channel_options>
    color-map:
      type: <ramp|interval|values>
      entries:
      - [color, entry_opacity, band_value, text_label]
    contrast-enhancement:
      mode: <normalize|histogram>
      gamma: <expression>
```

where:

Property	Required?	Description	Default value
opacity	No	Opacity of the entire display. Valid values are a decimal between 0 (completely transparent) and 1 (completely opaque).	1
channels	No	Selects the band(s) to display and the display method.	N/A
gray	No	Display a single band as a grayscale image. Cannot be used with red, green, and blue. The <channel_options> can be the band name or a mapping containing name: and contrast-enhancement: (optional).	1
red	No	Display three bands as an RGB image. Must be used with green, and blue. Cannot be used with gray. The <channel_options> can be the band name or a mapping containing name: and contrast-enhancement: (optional).	1
green	No	Display three bands as an RGB image. Must be used with red, and blue. Cannot be used with gray. The <channel_options> can be the band name or a mapping containing name: and contrast-enhancement: (optional).	2
blue	No	Display three bands as an RGB image. Must be used with red, and green. Cannot be used with gray. The <channel_options> can be the band name or a mapping containing name: and contrast-enhancement: (optional). See examples below.	3
color-map	No	Creates a mapping of colors to grid values. Can only be used with a single band.	N/A
type	No	Type of color mapping. Options are ramp, an interpolated list of values; interval, a non-interpolated list of values; and values, where values need to match exactly to be drawn.	ramp
entries	No	Values for the color mapping. Syntax is a list of tuples.	N/A
color	Yes	Color for the particular color map entry. Value is a standard color value.	N/A
entry_opacity	Yes	Opacity of the particular color map entry. Valid values are a decimal between 0 (completely transparent) and 1 (completely opaque).	N/A
band_value	Yes	Grid value to use for the particular color map entry. Values are data-dependent. Behavior at or around this value is determined by the color ramp type.	N/A
text_label	No	Label for the particular color map entry	Blank
contrast-enhancement	No	Modifies the contrast of the display	N/A
mode	No	Type of contrast enhancement. Options are normalize (stretches contrast so that the smallest and largest values are set to black and white, respectively) or histogram (produces equal number of content in the image at each brightness level).	normalize
gamma	No	Multiplier value for contrast adjustment. A value greater than 1 will increase darkness, while a value less than 1 will decrease darkness.	1

Examples

Enhanced contrast

This example takes a given raster and lightens the output by a factor of 2:

```
symbolizers:
- raster:
    contrast-enhancement:
      gamma: 0.5
```

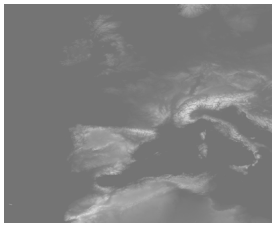


Fig. 6.193: Lightened image

Normalized output

This example takes a given raster and adjusts the contrast so that the smallest values are darkest and the highest values are lightest:

```
symbolizers:
- raster:
    contrast-enhancement:
      mode: normalize
```

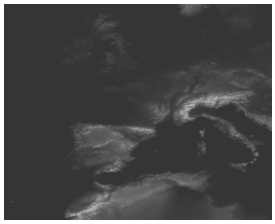


Fig. 6.194: Normalized image

Band selection

This example takes a raster with multiple bands and outputs band 2 as a grayscale image (This could be used to select a single band in a multi-band image to use with `color-map`):

```
name: raster
feature-styles:
- name: name
```

```
rules:
- symbolizers:
  - raster:
      opacity: 1.0
      channels:
        gray: 2
```

Fig. 6.195: Grayscale band selection

Band selection with contrast

This example takes an RGB raster, doubles the intensity of the red, and normalizes the green band:

```
name: raster
feature-styles:
- name: name
  rules:
  - symbolizers:
    - raster:
        channels:
          red:
            name: 1
            contrast-enhancement:
              gamma: .5
          green:
            name: 2
            contrast-enhancement:
              mode: normalize
          blue:
            name: 3
```

Fig. 6.196: Band selection with contrast enhancement

Color ramp

This example shows a color ramp from red to green to blue, with raster band values from 0-200:

```
symbolizers:
- raster:
    color-map:
      type: ramp
      entries:
      - ['#FF0000', 1, 0, red]
      - ['#00FF00', 1, 100, green]
      - ['#0000FF', 1, 200, blue]
```

In this example, the grid values will have the following colors applied:

- Less than or equal to 0 will have an output color of **solid red**
- Between 0 and 100 will have an output color **interpolated between red and green**
- Between 100 and 200 will have an output color **interpolated between green and blue**
- Greater than 200 will have an output color of **solid blue**

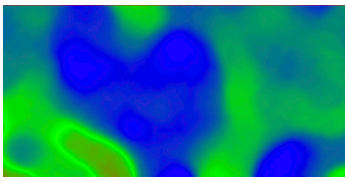


Fig. 6.197: Color map with ramp

Color intervals

The same example as above, but with the `color-map` type set to `intervals`:

```
symbolizers:
- raster:
    color-map:
      type: intervals
      entries:
      - ['#FF0000', 1, 0, red]
      - ['#00FF00', 1, 100, green]
      - ['#0000FF', 1, 200, blue]
```

In this example, the grid values will have the following colors applied:

- Less than or equal to 0 will have an output color of **solid red**
- Between 0 and 100 will have an output color of **solid green**
- Between 100 and 200 will have an output color of **solid blue**

- Greater than 200 will **not be colored** at all (transparent)

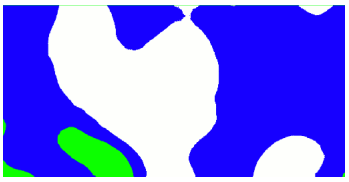


Fig. 6.198: Color map with intervals

Color values

The same example as above, but with the `color-map` type set to `values`:

```
symbolizers:
- raster:
    color-map:
      type: values
      entries:
      - ['#FF0000', 1, 0, red]
      - ['#00FF00', 1, 100, green]
      - ['#0000FF', 1, 200, blue]
```

In this example, the grid values will have the following colors applied:

- Equal to 0 will have an output color of **solid red**
- Equal to 100 will have an output color of **solid green**
- Equal to 200 will have an output color of **solid blue**

Any other values (even those in between the above values) will not be colored at all.

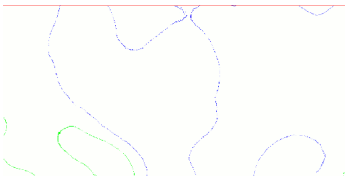


Fig. 6.199: Color map with values

Text symbolizer

The text symbolizer styles labels of vector features. Labels can consist of text strings and/or graphics.

Syntax

The full syntax of a text symbolizer is:

```
symbolizers:
- text:
    fill-color: <color>
    fill-opacity: <expression>
    fill-graphic:
      <graphic_options>
    stroke-graphic:
      <graphic_options>
    stroke-graphic-fill:
      <graphic_options>
    label: <expression>
    font-family: <expression>
    font-size: <expression>
    font-style: <expression>
    font-weight: <expression>
    placement: <point|line>
    offset: <expression>
    anchor: <tuple>
    displacement: <tuple>
    rotation: <expression>
    priority: <expression>
    halo:
      radius: <expression>
      fill-color: <color>
      fill-opacity: <expression>
      fill-graphic:
        <graphic_options>
    graphic:
      symbols:
        <graphic_options>
      size: <expression>
      opacity: <expression>
      rotation: <expression>
    geometry: <expression>
    uom: <text>
    x-composite-base: <boolean>
    x-composite: <text>
    x-allowOverruns: <boolean>
    x-autoWrap: <expression>
    x-conflictResolution: <boolean>
    x-followLine: <boolean>
    x-forceLeftToRight: <boolean>
    x-goodnessOfFit: <expression>
    x-graphic-margin: <expression>
    x-graphic-resize: <none|proportional|stretch>
    x-group: <boolean>
    x-labelAllGroup: <boolean>
    x-repeat: <expression>
    x-maxAngleDelta: <expression>
    x-maxDisplacement: <expression>
    x-minGroupDistance: <expression>
    x-partials: <boolean>
    x-polygonAlign: <boolean>
    x-spaceAround: <expression>
    x-underlineText: <boolean>
    x-strikethroughText: <boolean>
    x-charSpacing: <expression>
    x-wordSpacing: <expression>
```

where:

Property	Required?	Description	Default value
`fill-color`	No	Color of inside of the label.	`'#808080'` (gray)
`fill-opacity`	No	Opacity of fill of the label text. Valid values are a decimal value between `0` (completely transparent) and `1` (completely opaque).	`1`
`fill-graphic`	No	A design to be used for the fill of the text label. Can either be a mark consisting of a common shape or a URL that points to a graphic. The `<graphic_options>` should consist of a mapping containing `symbols:` followed by an `external:` or `mark:`, with appropriate parameters as detailed in the *Point symbolizer* section.	None
`stroke-graphic`	No	A design or pattern to be used along the stroke around the label text. the output will always be a linear repeating pattern, and as such is not tied to the value of `stroke-width`. Can either be a mark consisting of a common shape or a URL that points to a graphic. The `<graphic_options>` should consist of a mapping containing `symbols:` followed by an `external:` or `mark:`, with appropriate parameters. Cannot be used with `stroke-graphic-fill`.	N/A
`stroke-graphic-fill`	No	A design or pattern to be used for the fill of the stroke around the label text. This area that is to be filled is tied directly to the value of `stroke-width`. Can either be a mark consisting of a common shape or a URL that points to a graphic. The `<graphic_options>` should consist of a mapping containing `symbols:` followed by an `external:` or `mark:`, with appropriate parameters as detailed in the *Point symbolizer* section. Cannot be used with `stroke-graphic`.	N/A

Property	Required?	Description	Default value
`label`	Yes	Text to display. Often taken from an attribute but any valid expression that constructs a string will do.	N/A
`font-family`	No	Type of font to be used for the label. Options are system dependent; the full list of fonts available can be found via the GeoServer Server Status page.	`serif`
`font-size`	No	Size of the font.	`10`
`font-style`	No	Style of the font. Options are `normal`, `italic`, and `oblique`.	`normal`
`font-weight`	No	Weight of the font. Options are `normal` and `bold`.	`normal`
`placement`	No	Determines whether the label is to be drawn derived from a `point` or a `line`.	`point`
`offset`	No	Value (in pixels) for moving the drawn label relative to the location of the feature. A positive value will shift the label in the direction of its top, while a negative value will shift the label in the direction of its bottom. Only valid for when `type` is set to `line`.	`0`
`anchor`	No	Specify the center of the symbol relative to the feature location (centroid for lines and polygons). Value is an `[x,y]` tuple with decimal values from 0-1, with `[0,0]` meaning that the symbol is anchored to the bottom left of the label, and `[1,1]` meaning anchored to the top right of the label.	`[0,0]`
`displacement`	No	Specifies a distance (in pixels) to which to move the label relative to the feature. Value is an `[x,y]` tuple with values expressed in pixels, so [10,5] will displace the label 10 pixels to the right and 5 pixels up. Only valid for when `type` is set to `point`.	`[0,0]`
`rotation`	No	Value (in degrees) or rotation of the label. Larger values increase counter-clockwise rotation. A value of `180` will make the label upside-down. Only valid for when `type` is set to `point`.	`0`
`priority`	No	The priority used when choosing which labels to display during conflict resolution. Higher priority values take precedence over lower priority values.	1000
`halo`	No	Creates a shaded area around the label for easier legibility	No halo
`radius`	No	Size (in pixels) of the halo	`1`
`fill-color`	No	Color of the halo	`'#808080'`
`fill-opacity`	No	Specifies the level of transparency for the halo. Value of `0` means entirely transparent, while `1` means entirely opaque.	`1`

The following properties allow for a graphic to be displayed in addition to just a label. This is used when drawing "shields" (text over top of a graphic) such as in road signs.

Property	Required?	Description	Default value
`graphic`	No	Specifies whether a graphic is to be drawn for the label.	N/A (no graphic)
`symbols`	No	The details of the graphic. Consists of an `external:` or `mark:` section, with appropriate parameters as detailed in the *Point symbolizer* section.	N/A
`size`	No	Size of the graphic in pixels. If the aspect ratio is not 1:1 (square), will apply to the *height* of the graphic only, with the width scaled proportionally.	`16`
`opacity`	No	Specifies the level of transparency for the graphic. Value of `0` means entirely transparent, while `1` means entirely opaque.	`1`
`rotation`	No	Value (in degrees) or rotation of the graphic. Larger values increase counter-clockwise rotation. A value of `180` will make the graphic upside-down.	`0`

Property	Required?	Description	Default value
`geometry`	No	Specifies which attribute to use as the geometry.	First geometry attribute found (usually named `geom` or `the_geom`)
`uom`	No	Unit of measure used for width calculations	pixel

The following properties are equivalent to SLD "vendor options".

Property	Required?	Description	Default value
`x-allowOverruns`	No	Allows labels on lines to move slightly beyond the beginning or end of the line.	`true`
`x-autoWrap`	No	The number of pixels beyond which a label will be wrapped over multiple lines. Cannot use with `x-followLine`.	0
`x-conflictResolution`	No	Enables conflict resolution, meaning no two labels will be allowed to overlap. Without conflict resolution, symbolizers can overlap with other labels.	`true`
`x-followLine`	No	On linear geometries, the label will follow the shape of the current line, as opposed to being drawn at a tangent. Will override	`false`
`x-forceLeftToRight`	No	Forces labels to a readable orientation, otherwise will follow the line orientation, possibly making the label look upside-down. This setting is useful when using symbol fonts to add direction markers along a line.	`false`
`x-goodnessOfFit`	No	Percentage (expressed as a decimal between 0-1) of the label that must fit inside the geometry to permit the label to be drawn. Works only on polygon features.	0.5
`x-graphic-margin`	No	Number of pixels between the stretched graphic and the text. Only applies when `x-graphic-resize` is set to `stretch` or `proportional`.	0
`x-graphic-resize`	No	Allows for stretching the graphic underneath a label to fit the label size. Options are `none`, `stretch` or `proportional`. Used in conjunction with `x-graphic-margin`..	`none`
`x-group`	No	Geometries with identical labels will be considered a single entity to be labeled. Used to control repeated labels.	`false`
`x-labelAllGroup`	No	Used in conjunction with `x-group`. When `true` all items in a group are labeled. When `false`, only the largest geometry in the group is labeled. Valid for lines only.	`false`
`x-repeat`	No	Desired distance (in pixels) between labels drawn on a group. If zero, only one label will be drawn. Used in conjunction with `x-group`. Valid for lines only.	0
`x-maxAngleDelta`	No	Maximum allowed angle (in degrees) between two characters in a curved label. Used in conjunction with `x-followLine`. Values higher than `30` may cause loss of legibility of the label.	22.5
`x-maxDisplacement`	No	Distance (in pixels) a label can be displaced from its natural position in an attempt to eliminate conflict with other labels.	0
`x-minGroupDistance`	No	Minimum distance (in pixels) between two labels in the same label group. Used in conjunction with `displacement` or `repeat` to avoid having two labels too close to each other	No minimum distance
`x-partials`	No	Will display partial labels (truncated on the border of the display area).	`false`
`x-polygonAlign`	No	Overrides manual rotation to align label rotation automatically. Valid for polygons only.	`false`
`x-spaceAround`	No	Minimum distance (in pixels) between two labels. A negative value specifies the maximum overlap between two labels.	0
`x-underlineText`	No	Instruct the renderer to underline labels.	`false`

Property	Required?	Description	Default value
`x-composite`	No	Allows for both alpha compositing and color blending options between symbolizers.	N/A
`x-composite-base`	No	Allows the rendering engine to use the symbolizer mapping to define a "base" buffer for subsequent compositing and blending using `x-composite`. See the section on *Feature Styles* for more details.	`false`

Examples

Basic label

Text symbolizers are used to draw labels on objects. The label text is usually linked to some attribute of the layer. Font options are available in the `font-family`, `font-size`, `font-style`, and `font-weight` properties. The following example draws a label using the `name` attribute of the layer, and with a SansSerif font of size 12, gray color, bold and italic:

```
feature-styles:
- name: name
  rules:
  - title: fill-graphic
    symbolizers:
    - text:
        label: ${name}
        fill-color: '#555555'
        font-family: SansSerif
        font-size: 12
        font-style: italic
        font-weight: bold
```

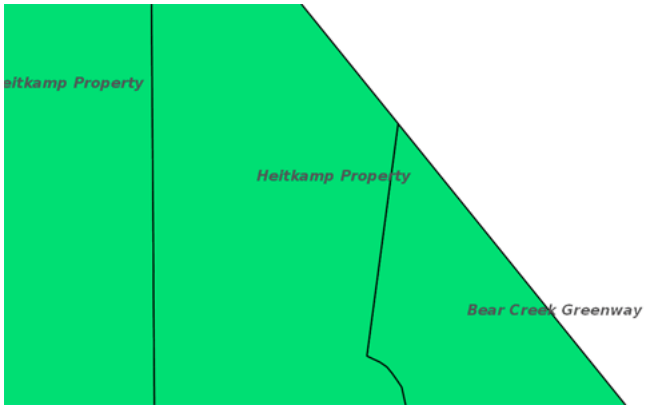


Fig. 6.200: Basic label

Label with wrap

Wrapping long labels can improve how well they fit on maps. This can be accomplished using the `x-autoWrap` property. This exam-

ple wraps lines longer than 70 pixels:

```
feature-styles:
- name: name
  rules:
  - symbolizers:
    - polygon:
        stroke-width: 1
        fill-color: '#00DD77'
    - text:
        label: ${name}
        font-size: 12
        x-autoWrap: 70
        x-maxDisplacement: 100
        anchor: [0.5,-1]
```

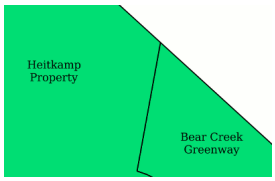


Fig. 6.201: Label with wrap

Label with halo

Surrounding labels with a halo will allow them to be visible even on complex maps with various background features. This can be accomplished using the `halo` family of properties. This example surrounds the label in a partially transparent white halo of radius 2:

```
feature-styles:
- name: name
  rules:
  - symbolizers:
    - polygon:
        stroke-width: 1
        fill-color: '#00DD77'
    - text:
        label: ${name}
        font-size: 12
        x-autoWrap: 70
        x-maxDisplacement: 100
        halo:
           radius: 2
           fill-color: '#FFFFFF'
           fill-opacity: 0.8
        anchor: [0.5,-1]
```

Grouped labels

Grouping and other properties can be used to better control where labels are placed. The `x-group` option combines all labels with

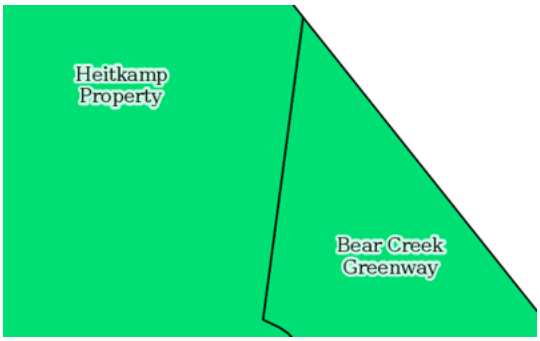


Fig. 6.202: Label with halo

identical text into a single label. This can be useful to show only a single label for a street rather than having a label on every block of the street. The `x-goodnesOfFit` option determines whether or not to draw labels based on how well they fit into the available space. The `x-maxDisplacement` option determines the maximum distance a label can be moved to avoid overlaps.

The following example uses `x-group` to ensure only one label is drawn for each feature, and sets `x-goodnesOfFit` to zero so that labels will be drawn even if they have a poor fit:

```
feature-styles:
- name: name
  rules:
  - title: fill-graphic
    symbolizers:
    - text:
        label: ${name}
        fill-color: '#555555'
        font-family: SansSerif
        font-size: 12
        font-style: italic
        font-weight: bold
        x-group: true
        x-goodnessOfFit: 0.0
        x-maxDisplacement: 400
```

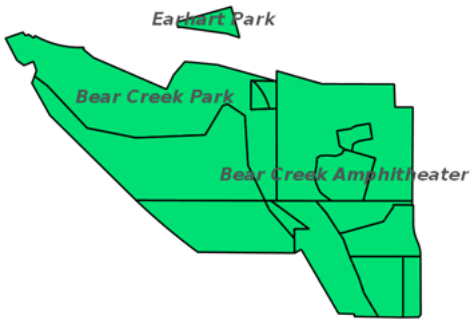


Fig. 6.203: Grouped labels

Labels following lines

In order to have a label follow a line (and not be drawn tangent to a line), the `x-followLine` option can be set. Other properties can be used in conjunction with this to achieve the best visual result. The following example has street names following the line of the street, with a maximum angle of 90 degrees, repeating every 150 pixels:

```
feature-styles:
- rules:
  - symbolizers:
    - line:
        stroke-color: '#EDEDFF'
        stroke-width: 10
    - text:
        label: name
        x-followLine: true
        x-maxAngleDelta: 90
        x-maxDisplacement: 400
        x-repeat: 150
```

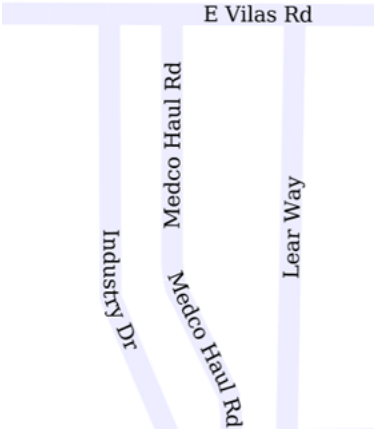


Fig. 6.204: Labels following lines

Labels avoiding obstacles

The `x-labelObstacle` option is used to mark a different symbolizer as an obstacle that labels should avoid. This example draws labels and points on a line geometry, and also uses a point symbolizer to draw the vertices of the lines as points. It is those points which are set to be treated as obstacles to be avoided:

```
feature-styles:
- rules:
  - symbolizers:
      - line:
          stroke-color: '#00BBDD'
          stroke-width: 10
- rules:
```

```
- symbolizers:
    - point:
        geometry: ${vertices(the_geom)}
        x-labelObstacle: true
        symbols:
        - mark:
            shape: circle
            stroke-color: '#000000'
            fill-color: '#007777'
    - text:
        label: ${streetname}
        x-maxDisplacement: 400
        x-followLine: true
```

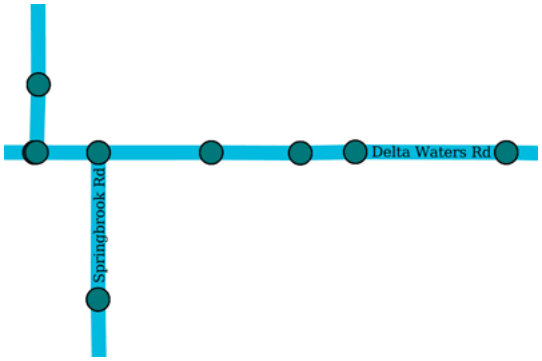


Fig. 6.205: Labels avoiding obstacles

Road Shields

The `graphic` option is used to display a symbol behind a label. A common use for this is to display "highway shields" behind road numbers. This example uses a circle `shape` to draw state shields, and an `external` image to draw interstate shields, then draws road names on top. The `x-graphic-resize` and `x-graphic-margin` options are used to resize the graphics to fit the label text:

```
feature-styles:
- name: state
  rules:
  - filter: ${level ilike 'State'}
    symbolizers:
    - line:
        stroke-color: '#AAEE00'
        stroke-width: 4
        stroke-linecap: round
    - text:
        label: ${name}
        anchor: [0.5, 0.5]
        fill-color: black
        font-family: SansSerif
        font-weight: bold
        font-size: 8
        x-graphic-resize: stretch
```

```
            x-graphic-margin: 6
            graphic:
              symbols:
              - mark:
                  shape: circle
                  fill-color: white
                  stroke-color: black
- name: interstate
  rules:
  - filter: ${level ilike 'Interstate'}
    symbolizers:
    - line:
        stroke-color: '#99CC00'
        stroke-width: 6
        stroke-linecap: round
    - text:
        label: ${name}
        anchor: [0.5, 0.5]
        fill-color: white
        font-family: SansSerif
        font-weight: bold
        font-size: 8
        x-graphic-resize: stretch
        x-graphic-margin: 6
        graphic:
          symbols:
          - external:
              url: interstate.png
              format: image/png
```

Fig. 6.206: Road Shields

Scale and zoom

It is common for different *rules* to be applied at different zoom levels on a web map.

For example, on a roads layer, you would not not want to display every single road when viewing the whole world. Or perhaps you may wish to styles the same features differently depending on the zoom level. For example: a cities layer styled using points at low zoom levels (when "zoomed out") and with polygon borders at higher zoom levels ("zoomed in").

YSLD allows rules to be applied depending on the the scale or zoom level. You can specify by scale, or you can define zoom levels in terms of scales and specify by zoom level.

Warning: Be aware that scales for a layer (where a style is applied) may interact differently when the layer is contained in a map, if the map has a different coordinate reference system from the layer.

Scale syntax

The syntax for using a scale conditional parameter in a rule is:

```
rules:
- ...
  scale: [<min>,<max>]
  ...
```

where:

Attribute	Required?	Description	Default value
`min`	Yes	The minimum scale (inclusive) for which the rule will be applied. Value is a number, either decimal or integer.	N/A
`max`	Yes	The maximum scale (exclusive) for which the rule will be applied. Value is a number, either decimal or integer.	N/A

Note: It is not possible to use an expression for any of these values.

Use the literal strings `min` and `max` to denote where there are no lower or upper scale boundaries. For example, to denote that the scale is anything less than some `<max>` value:

```
scale: [min,<max>]
```

To denote that the scale is anything greater than or equal to some `<min>` value:

```
scale: [<min>,max]
```

Note: In the above examples, `min` and `max` are always literals, entered exactly like that, while `<min>` and `<max>` would be replaced by actual scalar values.

If the scale parameter is omitted entirely, then the rule will apply at all scales.

Scale examples

Three rules, all applicable at different scales:

```
rule:
- name: large_scale
  scale: [min,100000]
```

```
    symbolizers:
    - line:
        stroke-width: 3
        stroke-color: '#0165CD'
  - name: medium_scale
    scale: [100000,200000]
    symbolizers:
    - line:
        stroke-width: 2
        stroke-color: '#0165CD'
  - name: small_scale
    scale: [200000,max]
    symbolizers:
    - line:
        stroke-width: 1
        stroke-color: '#0165CD'
```

This example will display lines with:

- A stroke width of 3 at scales less than 100,000 (`large_scale`)
- A stroke width of 2 at scales between 100,000 and 200,000 (`medium_scale`)
- A stroke width of 1 at scales greater than 200,000 (`small_scale`)

Given the rules above, the following arbitrary sample scales would map to the rules as follows:

Scale	Rule
50000	large_scale
100000	medium_scale
150000	medium_scale
200000	small_scale
300000	small_scale

Note the edge cases, since the `min` value is inclusive and the `max` value is exclusive.

Scientific notation for scales

To make comprehension easier and to lessen the chance of errors, scale values can be expressed in scientific notation.

So a scale of `500000000`, which is equal to 5 × 10^8 (a 5 with eight zeros), can be replaced by `5e8`.

Relationship between scale and zoom

When working with web maps, often it is more convenient to talk about zoom levels instead of scales. The relationship between zoom and scale is context dependent.

For example, for EPSG:4326 with world boundaries, zoom level 0 (completely zoomed out) corresponds to a scale of approximately

279,541,000 with each subsequent zoom level having half the scale value. For EPSG:3857 (Web Mercator) with world boundaries, zoom level 0 corresponds to a scale of approximately 559,082,000, again with each subsequent zoom level having half the scale value.

But since zoom levels are discrete (0, 1, 2, etc.) and scale levels are continuous, it's actually a range of scale levels that corresponds to a given zoom level.

For example, if you have a situation where a zoom level 0 corresponds to a scale of 1,000,000 (and each subsequent zoom level is half that scale, as is common), you can set the scale values of your rules to be:

- `scale: [750000,1500000]` (includes 1,000,000)
- `scale: [340000,750000]` (includes 500,000)
- `scale: [160000,340000]` (includes 250,000)
- `scale: [80000,160000]` (includes 125,000)
- etc.

Also be aware of the inverse relationship between scale and zoom; **as the zoom level increases, the scale decreases.**

Zoom syntax

In certain limited cases, it can be more useful to specify scales by way of zoom levels for predefined gridsets. These can be any predefined gridsets in GeoServer.

Inside a rule, the syntax for using zoom levels is:

```
rules:
- ...
  zoom: [<min>, <max>]
  ...
```

where:

Attribute	Required?	Description	Default value
`min`	Yes	The minimum zoom level for which the rule will be applied. Value is an integer.	N/A
`max`	Yes	The maximum zoom level for which the rule will be applied. Value is an integer.	N/A

Note: It is not possible to use an expression for any of these values.

As with scales, use the literal strings `min` and `max` to denote where there are no lower or upper scale boundaries. For example, to denote that the zoom level is anything less than some `<max>` value:

```
zoom: [min,<max>]
```

To denote that the zoom level is anything greater than or equal to some `<min>` value:

```
zoom: [<min>,max]
```

Note: In the above examples, `min` and `max` are always literals, entered exactly like that, while `<min>` and `<max>` would be replaced by actual scalar values.

The `scale` and `zoom` parameters should not be used together in a rule (but if used, `scale` takes priority over `zoom`).

Specifying a grid

While every web map can have zoom levels, the specific relationship between a zoom level and its scale is dependent on the gridset (spatial reference system, extent, etc.) used.

So when specifying zoom levels in YSLD, you should also specify the grid.

The `grid` parameter should remain at the top of the YSLD content, above any *Feature Styles* or *Rules*. The syntax is:

```
grid:
  name: <string>
```

where:

Property	Required?	Description	Default value
`name`	No	`WGS84`, `WebMercator`, or a name of a predefined gridset in GeoServer.	`WebMercator`

Note: As many web maps use "web mercator" (also known as EPSG:3857 or EPSG:900913), this is assumed to be the default if no `grid` is specified.

Warning: As multiple gridsets can contain the same SRS, we recommend naming custom gridsets by something other than the EPSG code.

Zoom examples

Default gridset

Given the default of web mercator (also known as EPSG:3857 or EPSG:900913), which requires no `grid` designation, this defines zoom levels as the following scale levels (rounded to the nearest whole number below):

Scale	Zoom level
559082264	0
279541132	1
139770566	2
69885283	3
34942641	4
17471321	5
8735660	6
4367830	7
2183915	8
<previous_scale> / 2	<previous_zoom> + 1

Named gridsets

For the existing gridset of `WGS84` (often known as `EPSG:4326`):

```
grid:
  name: WGS84
```

This defines zoom levels as the following scale levels (rounded to the nearest whole number below):

Scale	Zoom level
559082264	0
279541132	1
139770566	2
69885283	3
34942641	4
17471321	5
8735660	6
4367830	7
2183915	8
<previous_scale> / 2	<previous_zoom> + 1

Given a custom named gridset called `NYLongIslandFtUS`, defined by a CRS of EPSG:2263 and using its full extent:

```
grid:
  name: NYLongIslandFtUS
```

This defines zoom levels as the following (rounded to the nearest whole number below):

Scale	Zoom level
4381894	0
2190947	1
1095473	2
547736	3
273868	4
136934	5
68467	6
34234	7
17117	8
<previous_scale> / 2	<previous_zoom> + 1

Note: These scale values can be verified in GeoServer on the *Gridsets* page under the definition for the gridset:

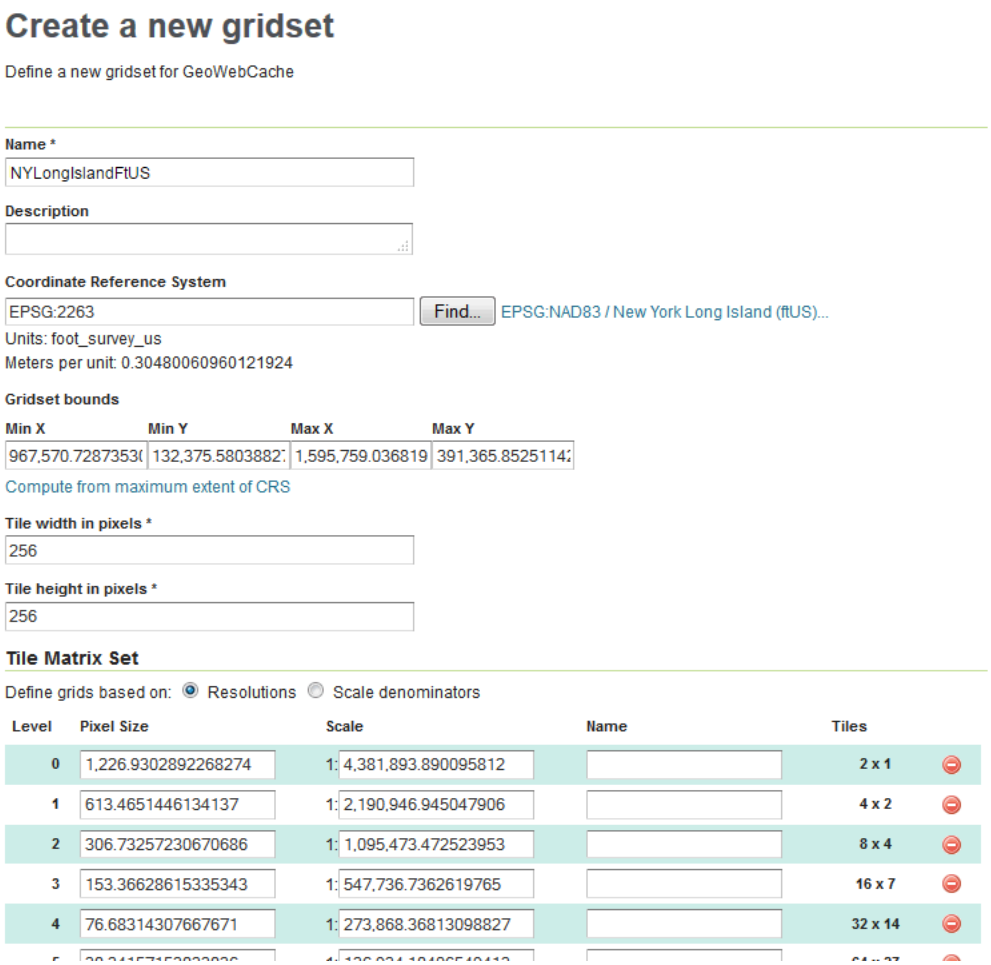


Fig. 6.207: Gridset defined in GeoServer

Specifically, note the *Scale* values under *Tile Matrix Set*.

Filters

Filters are predicates that allow rules to be applied selectively.

A filter can take a great many different forms.

Note: A scale is a type of filter, but is *discussed separately*.

Note: For more information, please see the GeoTools CQL documentation and GeoServer CQL tutorial.

Syntax

The basic syntax of a filter is:

```
rules:
  ...
  filter: ${<expression>}
  ...
```

where `<expression>` is any valid CQL/ECQL filter.

Note: Be aware that filters are applied to *rules* and so appear inside them, but outside of any *symbolizers*.

Types of filters

As mentioned above, the filter can be any valid construction made with CQL/ECQL (Extended/Contextual Query Language).

CQL is written using a familiar text-based syntax with strong similarities to SQL statements. One can think of a CQL expression as the "WHERE" clause of a SQL statement.

The following are all standard filter constructions:

Object comparison

This filter will test to see if a comparison to an attribute is true. It has the following form:

```
${<attribute> <operator> <value>}
```

where:

Attribute	Required?	Description	Default value
`<attribute>`	Yes	That to which something is going to be compared. Typically an **attribute name**. May be case sensitive.	N/A
`<operator>`	Yes	Method of comparison. Valid operators are `=`, `<`, `>`, `<=`, `>=`, `<>`, `LIKE`, `ILIKE`, `BETWEEN`, `IS NULL`, `IN`. `NOT` can be added to invert the comparison.	N/A
`<value>`	Yes	That which the `<attribute>` is being compared to. Typically a static value such as a string or scalar, though can be an expression that evaluates to a static value. Can include mathematical operators such as `+`, `-`, `*`, `/`. Use **single quotes** for strings, as double-quotes will be interpreted as an attribute name. Omit quotes for scalar values.	N/A

The following is a description of all available operators:

Operator	Description
`=`	Equals
`<`	Less than (non-inclusive)
`>`	Greater than (non-inclusive)
`<=`	Less than or equal to (inclusive)
`>=`	Greater than or equal to (inclusive)
`LIKE`	Fuzzy matching for strings and other non-numeric attributes. Add `%` for multi-character wildcards, and _ for single-character wildcards.
`ILIKE`	Case-insensitive version of `LIKE`
`BETWEEN`	Tests if a value that is between two given values
`IS NULL`	For testing against a `NULL` value
`IN`	Used when specifying a list. Must be contained in the list for the statement to be true.
`NOT`	Negates a boolean (true/false) condition. Can be used with an additional operator such as `NOT LIKE` or `NOT BETWEEN`.
`<>`	Not equal (used when comparing a string or numeric value only)

Note: These operators are not case sensitive, but are shown here in all caps for legibility and consistency.

Spatial filters

Filters can be spatial in nature. Any valid spatial construction in WKT (Well Known Text) can be used. Spatial filters include `INTERSECTS`, `DISJOINT`, `CONTAINS`, `WITHIN`, `TOUCHES`, `CROSSES`, `EQUALS`, `DWITHIN`, and `BBOX`. `NOT` can be added to negate the condition.

For more details about these spatial filters and their syntax, please see the GeoServer ECQL reference or uDig CQL reference.

Compound statements

The filter can be a combination of statements. A common case is testing if the value of an attribute is greater than one value but less than another.

The syntax for creating compound statements is to use standard Boolean notation such as `AND`, `OR`, and `NOT` along with relevant parentheses.

For example, a filter where both statements need to be true would be:

```
filter: ${<statement1> AND <statement2>}
```

A filter where either statement would need to be true would be:

```
filter: ${<statement1> OR <statement2>}
```

Larger filters can be built up in this way:

```
filter: ${(<statement1> OR␣
↪<statement2>) AND <statement3> OR NOT <statement4>}
```

In these examples, every `<statement>` is a valid filter.

In terms of precedence, `AND` is evaluated first, followed by `OR`, unless modified by parentheses. So, in the last example above, `(<statement1> OR <statement2>)` will be evaluated first, followed by the result of that `AND <statement3>`, and finally the result of that with `OR NOT <statement4>`.

Examples

Filter size based on an attribute

Filters are used to style different features of a layer based on certain conditions. The `ILIKE` operator is used to compare two strings (ignoring case) to see if they are similar. When using `LIKE` or `ILIKE`, the `%` character matches any number of letters (So `%hwy` matches any streetname ending in `hwy`). This example uses filters to distinguish between Highways, Roads, and other streets, and draw them using different colors and sizes:

```
feature-styles:
- rules:
  - filter: ${streetname ILIKE '%hwy'}
      symbolizers:
      - line:
          stroke-color: '#007799'
          stroke-width: 8
  - filter: ${streetname ILIKE '%rd'}
      symbolizers:
      - line:
          stroke-color: '#00AA00'
          stroke-width: 4
  - else: true
```

```
symbolizers:
- line:
    stroke-color: black
    stroke-width: 2
```

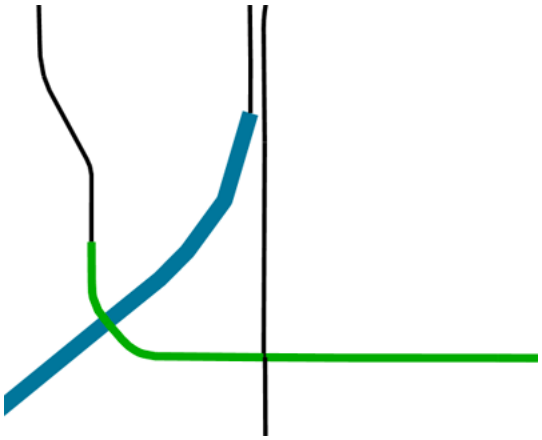


Fig. 6.208: Filter based on road types

Filter color based on attribute value

Filters can also be used to color a map based on attributes of the data. The following example uses the `YEARBLT` attribute to color different lots based on the year they were built. The `else` rule applies only if no other filter rule applies

Note: The Recode *function* can perform the same functionality in a more compact syntax.

```
name: Year Built Filter
feature-styles:
- rules:
  - filter: ${YEARBLT > 2000}
    symbolizers:
    - polygon:
        stroke-color: '#000000'
        stroke-width: 0.5
        fill-color: '#00FF00'
  - filter: ${YEARBLT > 1990 AND YEARBLT < 2000}
    symbolizers:
    - polygon:
        stroke-color: '#000000'
        stroke-width: 0.5
        fill-color: '#22DD00'
  - filter: ${YEARBLT > 1980 AND YEARBLT < 1990}
    symbolizers:
    - polygon:
        stroke-color: '#000000'
        stroke-width: 0.5
        fill-color: '#44BB00'
  - filter: ${YEARBLT > 1970 AND YEARBLT < 1980}
    symbolizers:
    - polygon:
```

```
            stroke-color: '#000000'
            stroke-width: 0.5
            fill-color: '#668800'
    - else: true
      symbolizers:
      - polygon:
            stroke-color: '#000000'
            stroke-width: 0.5
            fill-color: '#DD4400'
```

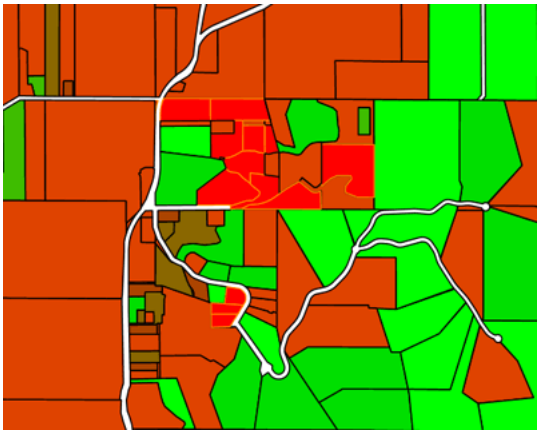


Fig. 6.209: Filter based on attribute value

Filter by bounding box

Spatial filters can be used to filter a layer based on its geometry. The `bbox` filter can be used to select features that are contained within a bounding box. This example colors polygons orange within the bounding box, and blue outside the bounding box:

```
name: Spatial Filter
feature-styles:
- name: name
  rules:
  -␣
↪filter: bbox(the_geom, -122.9, 42.36, -122.85, 42.28)
    symbolizers:
    - polygon:
        fill-color: '#99CC00'
  - else: true
    symbolizers:
    - polygon:
        fill-color: '#0099CC'
```

Filter by arbitrary geometries

Spatial filters can also be used to compare layer geometries against arbitrary geometries, not just bounding boxes. In this example, the `within` filter is used to select all buildings inside a triangular region defined using Well-Known Text (WKT) and color them green. All other features are colored blue:

Fig. 6.210: Detail of `bbox` filter

```
feature-styles:
- name: name
  rules:
  - filter:␣
↪within(the_geom, POLYGON ((-122.9075 42.3625, -122.
↪8225 42.3625, -122.8268 42.2803, -122.9075 42.3625)))
    symbolizers:
    - polygon:
        fill-color: '#00CC00'
  - else: true
    symbolizers:
    - polygon:
        fill-color: '#0099CC'
```

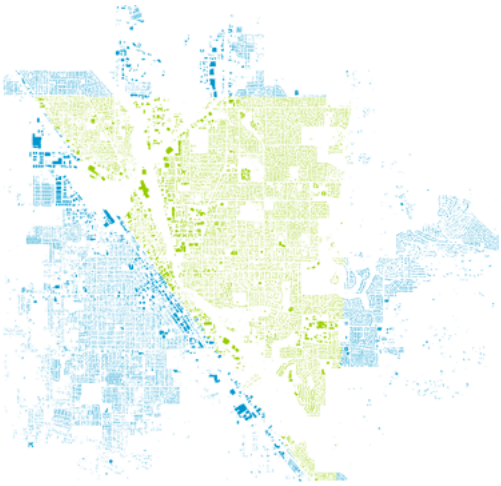


Fig. 6.211: Filter using `within`

Functions

Functions are additional operations that can be employed when calculating values for YSLD parameters. In most cases, a value for a parameter can be the output (result) of a function.

Functions can be used in most places in a style document.

Syntax

Functions aren't a parameter to itself, but instead are used as a part of the values of a parameter, or indeed in any expression. So the syntax is very general, for example:

```
<parameter>: ${<function>}
```

Functions are evaluated at rendering time, so the output is passed as the parameter value and then rendered accordingly.

List of functions

A reference list of functions can be found in the GeoServer User Manual and is also available in raw form in the GeoTools User Manual.

The functions can be broken up loosely into categories such as geometric, math, and string functions.

Theming functions

There are three important functions that are often easier to use for theming than using rules, and can vastly simplify your style documents: **Recode**, **Categorize**, and **Interpolate**.

Recode: Attribute values are directly mapped to styling properties:

```
recode(attribute,
↪value1,result1,value2,result2,value3,result3,...)
```

This is equivalent to creating multiple rules with similar filters:

```
rules:
- ...
  filter: ${attribute = value1}
  - ...
    <property>: result1
- ...
  filter: ${attribute = value2}
  - ...
    <property>: result2
- ...
  filter: ${attribute = value3}
  - ...
    <property>: result3
```

Categorize: Categories are defined using minimum and maximum ranges, and attribute values are sorted into the appropriate category:

```
categorize(attribute,category0,
↪value1,category1,value2,category2,...,belongsTo)
```

This would create a situation where the `attribute` value, if less than `value1` will be given the result of `category0`; if between `value1` and `value2`, will be given the result of `category1`; if between `value2` and `value3`, will be given the result of `category2`, etc. Values must be in ascending order.

The `belongsTo` argument is optional, and can be either `succeeding` or `preceding`. It defines which interval to use when the lookup value equals the `attribute` value. If the `attribute` value is equal to `value1` and `suceeding` is used, then the result will be `category1`. If `preceding` is used then the result will be `category0`. The default is `succeeding`.

This is equivalent to creating the following multiple rules:

```
rules:
- ...
  filter: ${attribute < value1}
  - ...
    <property>: category0
- ...
  filter: ${attribute >= value1 AND attribute < value2}
  - ...
    <property>: category1
- ...
  filter: ${attribute >= value2}
  - ...
    <property>: category2
```

Interpolate: Used to smoothly theme quantitative data by calculating a styling property based on an attribute value. This is similar to Categorize, except that the values are continuous and not discrete:

```
interpolate(attribute,
↪value1,entry1,value2,entry2,...,mode,method)
```

This would create a situation where the `attribute` value, if equal to `value1` will be given the result of `entry1`; if halfway between `value1` and `value2` will be given a result of halfway in between `entry1` and `entry2`; if three-quarters between `value1` and `value2` will be given a result of three-quarters in between `entry1` and `entry2`, etc.

The `mode` argument is optional, and can be either `linear`, `cosine`, or `cubic`. It defines the interpolation algorithm to use, and defaults to `linear`.

The `method` argument is optional, and can be either `numeric` or `color`. It determines whether `entry1`, `entry2`, ... are numbers or colors, and defaults to `numeric`.

There is no equivalent to this function in vector styling. The closest to this in raster styling is the color ramp.

The three theming functions can be neatly summarized by this table:

Function	Type of input	Type of output
Recode	Discrete	Discrete
Categorize	Continuous	Discrete
Interpolate	Continuous	Continuous

Examples

Display rotated arrows at line endpoints

The `startPoint(geom)` and `endPoint(geom)` functions take a geometry as an argument and returns the start and end points of the geometry respectively. The `startAngle(geom)` and `endAngle(geom)` functions take a geometry as an argument and return the angle of the line terminating at the start and end points of the geometry respectively. These functions can be used to display an arrow at the end of a line geometry, and rotate it to match the direction of the line:

```
feature-styles:
- rules:
  - symbolizers:
      - line:
          stroke-width: 1
      - point:
          geometry: ${endPoint(geom)}
          rotation: ${endAngle(geom)}
          size: 24
          symbols:
          - mark:
              shape: 'shape://carrow'
              fill-color: '#000000'
```

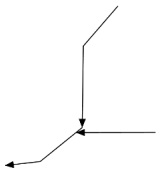


Fig. 6.212: Endpoint arrows

Drop shadow

The `offset(geom, x, y)` function takes a geometry and two values, and displaces the geometry by those values in the `x` and `y` directions. This can be used to create a drop-shadow effect:

```
feature-styles:
- name: shadow
  rules:
  - symbolizers:
```

```
      - polygon:
          stroke-width: 0.0
          fill-color: '#000000'
          fill-opacity: 0.75
          geometry: ${offset(geom, 0.0001, -0.0001)}
- name: fill
  rules:
  - symbolizers:
    - polygon:
      stroke-width: 0.0
      fill-color: '#00FFFF'
```

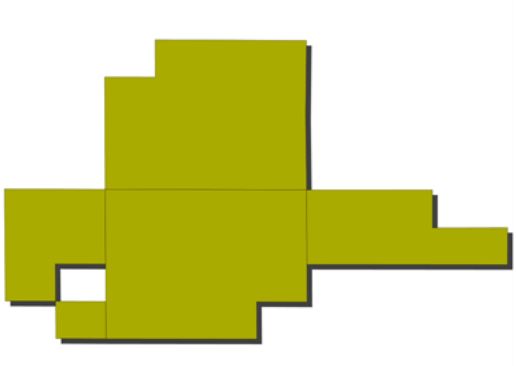


Fig. 6.213: Drop shadow

Different-colored outline

The `buffer(geom, buffer)` function takes a geometry and a value as arguments, and returns a polygon geometry with a boundary equal to the original geometry plus the value. This can be used to generate an extended outline filled with a different color, for example to style a shoreline:

```
feature-styles:
- name: shoreline
  rules:
  - polygon:
      fill-color: '#00BBFF'
      geometry: ${buffer(geom, 0.00025)}
- name: land
  rules:
  - polygon:
      fill-color: '#00DD00'
```

See also:

- convexHull(geom)
- octagonalEnvelope(geom)
- mincircle(geom)
- minrectangle(geom)

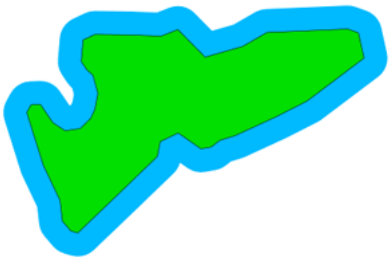


Fig. 6.214: Buffered outline

- minimumdiameter(geom)

Display vertices of a line

The `vertices(geom)` function takes a geometry and returns a collection of points representing the vertices of the geometry. This can be used to convert a polygon or line geometry into a point geometry:

```
point:
  geometry: vertices(geom)
```

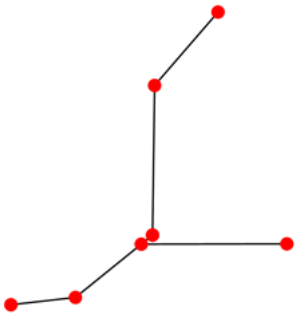


Fig. 6.215: Endpoint arrows

See also:

- boundary(geom)
- centroid(geom)

Angle between two points

The `atan2(x, y)` function calculates the arctangent of (y/x) and so is able to determine the angle (in radians) between two points. This function uses the signs of the x and y values to determine the computed angle, so it is preferable over

`atan()`. The `getX(point_geom)` and `getY(point_geom)` extracts the `x` and `y` ordinates from a geometry respectively, while `toDegrees(value)` converts from radians to degrees:

```
point:
  symbols:
  - mark:
      shape: triangle
  rotation: ${toDegrees(atan2(
  ␣
↪ getX(startPoint(the_geom))-getX(endPoint(the_geom)),
    getY(startPoint(the_
↪geom))-getY(endPoint(the_geom))))}
```

See also:

- sin(value)
- cos(value)
- tan(value)
- asin(value)
- acos(value)
- atan(value)
- toRadians(value)
- pi()

Scale objects based on a large range of values

The `log(value)` function returns the natural logarithm of the provided value. Use `log(value)/log(base)` to specify a different base.

For example, specifying `log(population)/log(2)` will make the output increase by 1 when the value of population doubles. This allows one to display relative sizes on a consistent scale while still being able to represent very small and very large populations:

```
point:
  symbols:
  - mark:
      shape: circle
  size: ${log(population)/log(2)}
```

See also:

- exp(val)
- pow(base,exponent)
- sqrt(val)

Combine several strings into one

The `Concatenate(string1, string2, ...)` function takes any number of strings and combines them to form a single string. This can be used to display more than one attribute within a single label:

```
text:
  label: ${Concatenate(name, ', ', population)}
```

Capitalize words

The `strCapitalize(string)` function takes a single string and capitalizes the first letter of each word in the string. This could be used to capitalize labels created from lower case text:

```
text:
  label: ${strCapitalize(name)}
```

See also:

- strToLowerCase(string)
- strToUpperCase(string)

Color based on discrete values

In certain cases, theming functions can be used in place of filters to produce similar output much more simply. For example, the Recode function can take an attribute and output a different value based on an attribute value. So instead of various filters, the entire constructions can be done in a single line. For example, this could be used to color different types of buildings:

```
feature-styles:
- name: name
  rules:
  - symbolizers:
    - polygon:
        fill-color:
          ${recode(zone,
          'I-L', '#FF7700',
          'I-H', '#BB6600',
          'C-H', '#0077BB',
          'C-R', '#00BBDD',
          'C-C', '#00DDFF',
          '', '#777777')}
```

In the above example, the attribute is `zone` , and then each subsequent pair consists of an attribute value followed by a color.

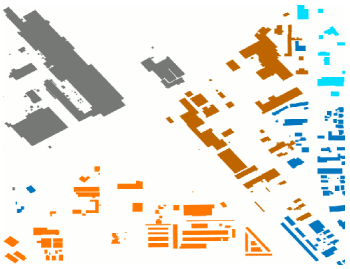


Fig. 6.216: Recode Function

Color based on categories

The Categorize function returns a different value depending on which range (category) an attribute value matches. This can also make a style much more simple by reducing the number of filters. This example uses `categorize` to color based on certain values of the `YEARBLT` attribute:

```
feature-styles:
- name: name
  rules:
  - symbolizers:
      - polygon:
          stroke-color: '#000000'
          stroke-width: 0.5
          fill-color:
            ${categorize(YEARBLT, '#DD4400',
            1950,'#AA4400',
            1960,'#886600',
            1970,'#668800',
            1980,'#44BB00',
            1990,'#22DD00',
            2000,'#00FF00')}
```

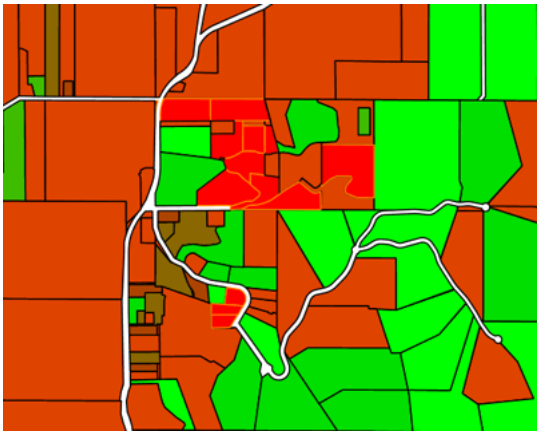


Fig. 6.217: Categorize Function

Choropleth map

The `interpolate` function can be used to create a continuous set of values by interpolating between attribute values. This can be used to create a choropleth map which shows different colors for regions based on some continuous attribute such as area or population:

```
feature-styles:
- name: name
  rules:
  - title: fill-graphic
    symbolizers:
    - polygon:
        stroke-width: 1
        fill-color: ${interpolate(PERSONS,
↪ 0.0, '#00FF00', 1e7,'#FF0000', 'color')}
```

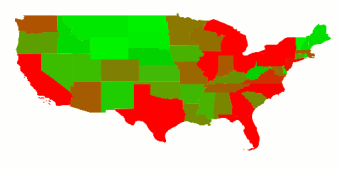


Fig. 6.218: Choropleth Map

Variables

Variables in YSLD that allow for a certain directive or block of directives to be defined by name and later reused. This can greatly simplify the styling document.

The two most-common use cases for using variables are:

- To create a more-friendly name for a value (such as using `myorange` instead of `#EE8000`)
- To define a block of directives to remove redundant content and to decrease file length

It is customary, but not required, to place all definitions at the very top of the YSLD file, above all *header information*, *feature styles*, or *rules*.

Syntax

Single value

The syntax for defining a variable as a single value is:

```
define: &variable <value>
```

where:

Attribute	Required?	Description	Default value
`define`	Yes	Starts the definition block.	N/A
`&variable`	Yes	The name of the variable being defined. The `&` is not part of the variable name.	N/A
`<value>`	Yes	A single value, such as `512` or `'#DD0000'`	N/A

The syntax for using this variable is to prepend the variable name with a `*`:

```
<directive>: *variable
```

This variable can be used in any place where its type is expected.

Directive block

The syntax for defining a variable as a content block is:

```
define: &varblock
  <directive>: <value>
  <directive>: <value>
  ...
  <block>:
  - <directive>: <value>
    <directive>: <value>
  ...
```

Any number of directives or blocks of directives can be inside the definition block. Moreover, any type of directive that is valid YSLD can be included in the definition, so long as the content block could be substituted for the variable without modification.

Note: It is also possible to have nested definitions.

The syntax for using this variable is to prepend the variable name with `<<: *`. For example:

```
<block>:
- <directive>: <value>
  <<: *varblock
```

The line that contains the variable will be replaced with the contents of the definition.

Examples

The following are all examples of variable substitution

Name a color:

```
define: &myorange '#EE8000'
```

```
stroke: *myorange
```

Reusable text string:

```
define: &rulename "This is my rule"
```

```
title: *rulename
```

Stroke style:

```
define: &strokestyle
  stroke: '#FF0000'
  stroke-width: 2
  stroke-opacity: 0.5
```

```
polygon:
  <<: *strokestyle
```

Transforms

YSLD allows for the use of rendering transformations. Rendering transformations are processes on the server that are executed inside the rendering pipeline, to allow for dynamic data transformations. In GeoServer, rendering transformations are typically exposed as WPS processes.

For example, one could create a style that applies to a point layer, and applies a Heatmap process as a rendering transformation, making the output a (raster) heatmap.

Because rendering transformations can change the geometry type, it is important to make sure that the *symbolizer* used matches the *output* of the rendering transformation, not the input. In the above heatmap example, the appropriate symbolizer would be a raster symbolizer, as the output of a heatmap is a raster.

Syntax

The full syntax for using a rendering transformation is:

```
feature-styles
  ...
  transform:
    name: <text>
    params: <options>
  rules:
    ...
```

where:

Property	Required?	Description	Default value
`name`	Yes	Full name of the rendering transform including any prefixes (such as `vec:Heatmap`)	N/A
`params`	Yes	All input parameters for the rendering transformation. Content will vary greatly based on the amount and type of parameters needed.	N/A

The values in the `params` options typically include values, strings, or attributes. However, it can be useful with a transformation to include environment parameters that concern the position and size of the map when it is rendered. For example, the following are common reserved environment parameters:

Environment parameter	Description
`env('wms_bbox')`	The bounding box of the request
`env('wms_width')`	The width of the request
`env('wms_height')`	The height of the request

With this in mind, the following `params` are assumed unless otherwise specified:

```
params:
  ...
  outputBBOX: ${env('wms_bbox')}
  outputWidth: ${env('wms_width')}
  outputHeight: ${env('wms_height')}
  ...
```

Note: Be aware that the transform happens *outside* of the *rules* and *symbolizers*, but inside the *feature styles*.

Examples

Heatmap

The following uses the `vec:Heatmap` process to convert a point layer to a heatmap raster:

```
title: Heatmap
feature-styles:
- transform:
    name: vec:Heatmap
    params:
      weightAttr: pop2000
      radiusPixels: 100
      pixelsPerCell: 10
  rules:
  - symbolizers:
    - raster:
        opacity: 0.6
        color-map:
```

```
                    type: ramp
                    entries:
                    - ['#FFFFFF',0,0.0,nodata]
                    - ['#4444FF',1,0.1,nodata]
                    - ['#FF0000',1,0.5,values]
                    - ['#FFFF00',1,1.0,values]
```

Point Stacker

The point stacker transform can be used to combine points that are close together. This transform acts on a point geometry layer, and combines any points that are within a single cell as specified by the `cellSize` parameter. The resulting geometry has attributes `geom` (the geometry), `count` (the number of features represented by this point) and `countUnique` (the number of unique features represented by this point). These attributes can be used to size and label the points based on how many points are combined together:

```
title: pointstacker
feature-styles:
- transform:
    name: vec:PointStacker
    params:
    cellSize: 100
  rules:
  - symbolizers:
    - point:
        size: ${8*sqrt(count)}
        symbols:
        - mark:
            shape: circle
            fill-color: '#EE0000'
  - filter: count > 1
    symbolizers:
    - text:
          fill-color: '#FFFFFF'
          font-family: Arial
          font-size: 10
          font-weight: bold
          label: ${count}
          placement:
              anchor: [0.5,0.75]
```

6.5.4 YSLD Cookbook

The YSLD Cookbook is a collection of YSLD "recipes" for creating various types of map styles. Wherever possible, each example is designed to show off a single YSLD feature so that code can be copied from the examples and adapted when creating YSLDs of your own. While not an exhaustive reference like the *YSLD reference* the YSLD cookbook is designed to be a practical reference, showing common style templates that are easy to understand.

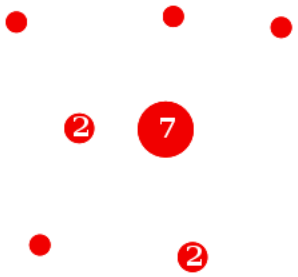


Fig. 6.219: Point stacker

The YSLD Cookbook is divided into four sections: the first three for each of the vector types (points, lines, and polygons) and the fourth section for rasters. Each example in every section contains a screenshot showing actual GeoServer WMS output, a snippet of the YSLD code for reference, and a link to download the full YSLD.

Each section uses data created especially for the YSLD Cookbook, with shapefiles for vector data and GeoTIFFs for raster data. The projection for data is EPSG:4326.

Data Type	Shapefile
Point	`ysld_cookbook_point.zip`
Line	`ysld_cookbook_line.zip`
Polygon	`ysld_cookbook_polygon.zip`
Raster	`ysld_cookbook_raster.zip`

Points

While points are seemingly the simplest type of shape, possessing only position and no other dimensions, there are many different ways that a point can be styled in YSLD.

Example points layer

The `points layer` used for the examples below contains name and population information for the major cities of a fictional country. For reference, the attribute table for the points in this layer is included below.

fid (Feature ID)	name (City name)	pop (Population)
point.1	Borfin	157860
point.2	Supox City	578231
point.3	Ruckis	98159
point.4	Thisland	34879
point.5	Synopolis	24567
point.6	San Glissando	76024
point.7	Detrania	205609

Download the points shapefile

Simple point

This example specifies points be styled as red circles with a diameter of 6 pixels.

Fig. 6.220: Simple point

Code

Download the "Simple point" YSLD

```
title: 'YSLD Cook Book: Simple Point'
feature-styles:
- name: name
  rules:
  - symbolizers:
    - point:
        size: 6
        symbols:
        - mark:
            shape: circle
            fill-color: '#FF0000'
```

Details

There is one rule in one feature style for this YSLD, which is the simplest possible situation. (All subsequent examples will contain one rule and one feature style unless otherwise specified.) Styling points is accomplished via the point symbolizer (**lines 6-11**). **Line 10** specifies the shape of the symbol to be a circle, with **line 11** determining the fill color to be red (`'#FF0000'`). **Line 7** sets the size (diameter) of the graphic to be 6 pixels.

Simple point with stroke

This example adds a stroke (or border) around the *Simple point*, with the stroke colored black and given a thickness of 2 pixels.

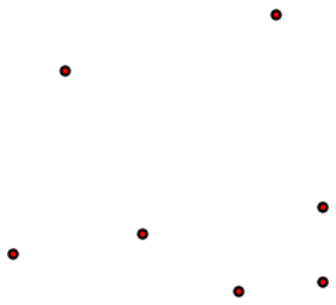


Fig. 6.221: Simple point with stroke

Code

Download the "Simple point with stroke" YSLD

```
1  title: 'YSLD Cook Book: Simple point with stroke'
2  feature-styles:
3  - name: name
4    rules:
5    - symbolizers:
6      - point:
7          size: 6
8          symbols:
9          - mark:
10             shape: circle
11             stroke-color: '#000000'
12             stroke-width: 2
13             fill-color: '#FF0000'
```

Details

This example is similar to the *Simple point* example. **Lines 11-12** specify the stroke, with **line 11** setting the color to black (`'#000000'`) and **line 12** setting the width to 2 pixels.

Rotated square

This example creates a square instead of a circle, colors it green, sizes it to 12 pixels, and rotates it by 45 degrees.

Fig. 6.222: Rotated square

Code

Download the "Rotated square" YSLD

```
1                                   title: 'YSLD Cook Book: Rotated square'
2                                   feature-styles:
3                                   - name: name
4                                     rules:
5                                     - symbolizers:
6                                       - point:
7                                           size: 12
8                                           rotation: 45
9                                           symbols:
10                                          - mark:
11                                              shape: square
12                                              fill-color: '#009900'
```

Details

In this example, **line 11** sets the shape to be a square, with **line 12** setting the color to a dark green (`009900`). **Line 7** sets the size of the square to be 12 pixels, and **line 8** sets the rotation to 45 degrees.

Transparent triangle

This example draws a triangle, creates a black stroke identical to the *Simple point with stroke* example, and sets the fill of the triangle to 20% opacity (mostly transparent).

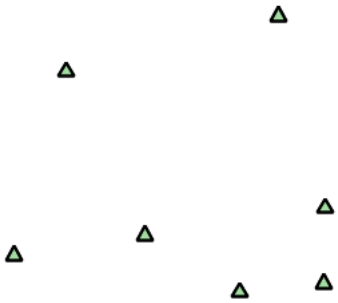


Fig. 6.223: Transparent triangle

Code

Download the "Transparent triangle" YSLD

```
1  title: 'YSLD Cook Book: Transparent triangle'
2  feature-styles:
3  - name: name
4    rules:
5    - symbolizers:
6      - point:
7          size: 12
8          symbols:
9          - mark:
10             shape: triangle
11             stroke-color: '#000000'
12             stroke-width: 2
13             fill-color: '#009900'
14             fill-opacity: 0.2
```

Details

In this example, **line 10** once again sets the shape, in this case to a triangle. **Line 13** sets the fill color to a dark green (`'#009900'`) and **line 14** sets the opacity to 0.2 (20% opaque). An opacity value of 1 means that the shape is drawn 100% opaque, while an opacity value of 0 means that the shape is drawn 0% opaque, or completely transparent. The value of 0.2 (20% opaque) means that the fill of the points partially takes on the color and style of whatever is drawn beneath it. In this example, since the background is white, the dark green looks lighter. Were the points imposed on a dark background,

the resulting color would be darker. **Lines 11-12** set the stroke color to black (`'#000000'`) and width to 2 pixels. Finally, **line 7** sets the size of the point to be 12 pixels in diameter.

Point as graphic

This example styles each point as a graphic instead of as a simple shape.

Fig. 6.224: Point as graphic

Code

Download the "Point as graphic" YSLD

```
1  title: 'YSLD Cook Book: Point as graphic'
2  feature-styles:
3  - name: name
4    rules:
5    - symbolizers:
6      - point:
7          size: 32
8          symbols:
9          - external:
10             url: smileyface.png
11             format: image/png
```

Details

This style uses a graphic instead of a simple shape to render the points. In YSLD, this is known as an `external`, to distinguish it from the commonly-used shapes such as squares and circles that are "internal" to the renderer. **Lines 9-11** specify the details of this graphic. **Line 10** sets the path and file name of the graphic, while **line 11** indicates the format (MIME type) of the graphic (image/png). In this example, the graphic is contained in the same directory as the YSLD, so no path information is necessary in **line 10**,

although a full URL could be used if desired. **Line 7** determines the size of the displayed graphic; this can be set independently of the dimensions of the graphic itself, although in this case they are the same (32 pixels). Should a graphic be rectangular, the `size` value will apply to the *height* of the graphic only, with the width scaled proportionally.

Fig. 6.225: Graphic used for points

Point with default label

This example shows a text label on the *Simple point* that displays the "name" attribute of the point. This is how a label will be displayed in the absence of any other customization.

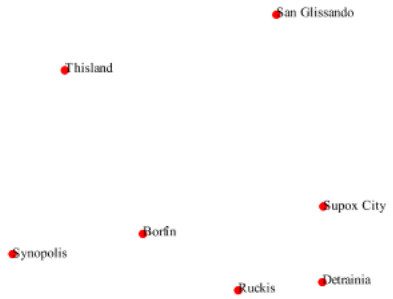


Fig. 6.226: Point with default label

Code

Download the "Point with default label" YSLD

```
title: 'YSLD Cook Book: Point with default label'
feature-styles:
- name: name
  rules:
  - symbolizers:
    - point:
        size: 6
        symbols:
        - mark:
            shape: circle
            fill-color: '#FF0000'
    - text:
        label: ${name}
        fill-color: '#000000'
```

```
15                                      font-family: Serif
16                                      font-size: 10
17                                      font-style: normal
18                                      font-weight: normal
19                                      placement: point
```

Details

Lines 2-11, which contain the point symbolizer, are identical to the *Simple point* example above. The label is set in the text symbolizer on **lines 12-19**. **Line 13** determines what text to display in the label, which in this case is the value of the "name" attribute. (Refer to the attribute table in the *Example points layer* section if necessary.) **Line 15** sets the text color. All other details about the label are set to the renderer default, which here is Times New Roman font, font color black, and font size of 10 pixels. The bottom left of the label is aligned with the center of the point.

Point with styled label

This example improves the label style from the *Point with default label* example by centering the label above the point and providing a different font name and size.

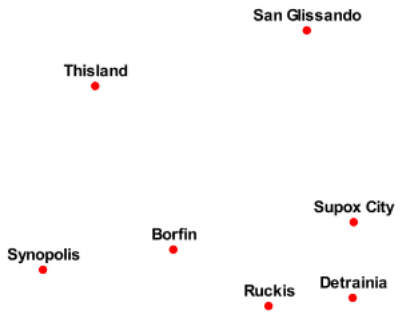


Fig. 6.227: Point with styled label

Code

Download the "Point with styled label" YSLD

```
1                                   title: 'YSLD Cook Book: Point with styled label'
2                                   feature-styles:
3                                   - name: name
4                                     rules:
5                                     - symbolizers:
6                                       - point:
```

```
                size: 6
                symbols:
                - mark:
                    shape: circle
                    fill-color: '#FF0000'
            - text:
                label: ${name}
                fill-color: '#000000'
                font-family: Arial
                font-size: 12
                font-style: normal
                font-weight: bold
                placement: point
                anchor: [0.5,0.0]
                displacement: [0,5]
```

Details

In this example, **lines 2-11** are identical to the *Simple point* example above. The `<TextSymbolizer>` on **lines 12-21** contains many more details about the label styling than the previous example, *Point with default label*. **Line 13** once again specifies the "name" attribute as text to display. **Lines 15-18** set the font information: **line 15** sets the font family to be "Arial", **line 16** sets the font size to 12, **line 17** sets the font style to "normal" (as opposed to "italic" or "oblique"), and **line 18** sets the font weight to "bold" (as opposed to "normal"). **Lines 19-21** determine the placement of the label relative to the point. The `anchor` (**line 20**) sets the point of intersection between the label and point, which here sets the point to be centered (0.5) horizontally axis and bottom aligned (0.0) vertically with the label. There is also `displacement` (**line 21**), which sets the offset of the label relative to the line, which in this case is 0 pixels horizontally and 5 pixels vertically . Finally, **line 14** sets the font color of the label to black (`'#000000'`).

The result is a centered bold label placed slightly above each point.

Point with rotated label

This example builds on the previous example, *Point with styled label*, by rotating the label by 45 degrees, positioning the labels farther away from the points, and changing the color of the label to purple.

Code

`Download the "Point with rotated label" YSLD`

```
title: 'YSLD Cook Book: Point with rotated label'
feature-styles:
- name: name
  rules:
  - symbolizers:
```

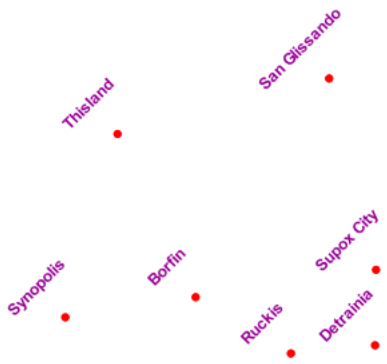


Fig. 6.228: Point with rotated label

```
6                                   - point:
7                                       size: 6
8                                       symbols:
9                                       - mark:
10                                          shape: circle
11                                          fill-color: '#FF0000'
12                                  - text:
13                                      label: ${name}
14                                      fill-color: '#990099'
15                                      font-family: Arial
16                                      font-size: 12
17                                      font-style: normal
18                                      font-weight: bold
19                                      placement: point
20                                      anchor: [0.5,0.0]
21                                      displacement: [0,25]
22                                      rotation: -45
```

Details

This example is similar to the *Point with styled label*, but there are three important differences. **Line 21** specifies 25 pixels of vertical displacement. **Line 22** specifies a rotation of "-45" or 45 degrees counter-clockwise. (Rotation values increase clockwise, which is why the value is negative.) Finally, **line 14** sets the font color to be a shade of purple (`'#99099'`).

Note that the displacement takes effect before the rotation during rendering, so in this example, the 25 pixel vertical displacement is itself rotated 45 degrees.

Attribute-based point

This example alters the size of the symbol based on the value of the population ("pop") attribute.

Fig. 6.229: Attribute-based point

Code

Download the "Attribute-based point" YSLD

```
title: 'YSLD Cook Book: Attribute-based point'
feature-styles:
- name: name
  rules:
  - name: SmallPop
    title: 1 to 50000
    filter: ${pop < '50000'}
    symbolizers:
    - point:
        size: 8
        symbols:
        - mark:
            shape: circle
            fill-color: '#0033CC'
  - name: MediumPop
    title: 50000 to 100000
    filter: ${pop >= '50000' AND pop < '100000'}
    symbolizers:
    - point:
        size: 12
        symbols:
        - mark:
            shape: circle
            fill-color: '#0033CC'
  - name: LargePop
    title: Greater than 100000
    filter: ${pop >= '100000'}
    symbolizers:
    - point:
        size: 16
        symbols:
        - mark:
            shape: circle
            fill-color: '#0033CC'
```

Details

Note: Refer to the *Example points layer* to see the attributes for this data. This example has eschewed labels in order to simplify the style, but you can refer to the example *Point with styled label* to see which attributes correspond to which points.

This style contains three rules. Each rule varies the style based on the value of the population ("pop") attribute for each point, with smaller values yielding a smaller circle, and larger values yielding a larger circle.

The three rules are designed as follows:

Rule order	Rule name	Population (`pop`)	Size
1	SmallPop	Less than 50,000	8
2	MediumPop	50,000 to 100,000	12
3	LargePop	Greater than 100,000	16

The order of the rules does not matter in this case, since each shape is only rendered by a single rule.

The first rule, on **lines 5-14**, specifies the styling of those points whose population attribute is less than 50,000. **Line 7** sets this filter, denoting the attribute ("pop") to be "less than" the value of 50,000. The symbol is a circle (**line 13**), the color is dark blue (`'#0033CC'`, on **line 15**), and the size is 8 pixels in diameter (**line 18**).

The second rule, on **lines 15-24**, specifies a style for points whose population attribute is greater than or equal to 50,000 and less than 100,000. The population filter is set on **line 17**. This filter specifies two criteria instead of one: a "greater than or equal to" and a "less than" filter. These criteria are joined by `AND`, which mandates that both filters need to be true for the rule to be applicable. The size of the graphic is set to 12 pixels on **line 20**. All other styling directives are identical to the first rule.

The third rule, on **lines 25-34**, specifies a style for points whose population attribute is greater than or equal to 100,000. The population filter is set on **line 27**, and the only other difference is the size of the circle, which in this rule (**line 30**) is 16 pixels.

The result of this style is that cities with larger populations have larger points.

Zoom-based point

This example alters the style of the points at different zoom levels.

Code

```
Download the "Zoom-based point" YSLD
```

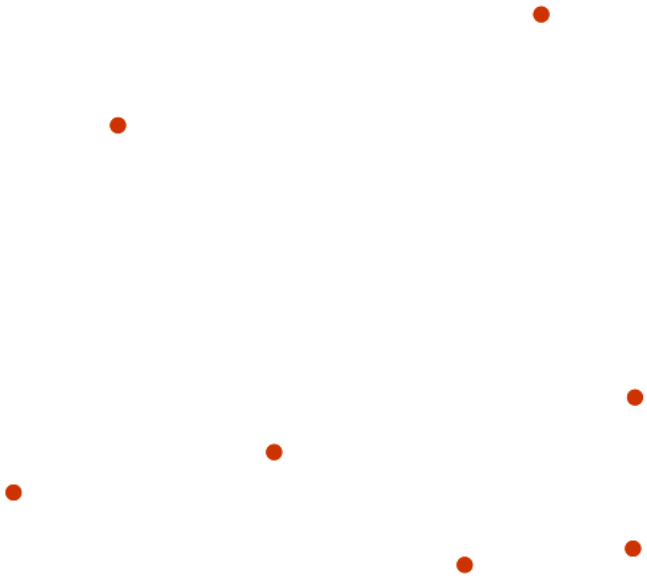


Fig. 6.230: Zoom-based point: Zoomed in

Fig. 6.231: Zoom-based point: Partially zoomed

Fig. 6.232: Zoom-based point: Zoomed out

```
title: 'YSLD Cook Book: Zoom-based point'
feature-styles:
- name: name
  rules:
  - name: Large
    scale: [min,1.6e8]
    symbolizers:
    - point:
        size: 12
        symbols:
        - mark:
            shape: circle
            fill-color: '#CC3300'
  - name: Medium
    scale: [1.6e8,3.2e8]
    symbolizers:
    - point:
        size: 8
        symbols:
        - mark:
            shape: circle
            fill-color: '#CC3300'
  - name: Small
    scale: [3.2e8,max]
    symbolizers:
    - point:
        size: 4
        symbols:
        - mark:
            shape: circle
            fill-color: '#CC3300'
```

Details

It is often desirable to make shapes larger at higher zoom levels when creating a natural-looking map. This example styles the points to vary in size based on the zoom level (or more accurately, scale denominator). Scale denominators refer to the scale of the map. A

scale denominator of 10,000 means the map has a scale of 1:10,000 in the units of the map projection.

Note: Determining the appropriate scale denominators (zoom levels) to use is beyond the scope of this example.

This style contains three rules. The three rules are designed as follows:

Rule order	Rule name	Scale denominator	Point size
1	Large	1:160,000,000 or less	12
2	Medium	1:160,000,000 to 1:320,000,000	8
3	Small	Greater than 1:320,000,000	4

The order of these rules does not matter since the scales denominated in each rule do not overlap.

The first rule (**lines 5-13**) is for the smallest scale denominator, corresponding to when the view is "zoomed in". The scale rule is set on **line 6**, so that the rule will apply to any map with a scale denominator of 160,000,000 or less. The rule draws a circle (**line 12**), colored red (`#CC3300` on **line 13**) with a size of 12 pixels (**line 9**).

The second rule (**lines 14-22**) is the intermediate scale denominator, corresponding to when the view is "partially zoomed". The scale rules is set on **line 15**, so that the rule will apply to any map with a scale denominator between 160,000,000 and 320,000,000. (The lower bound is inclusive and the upper bound is exclusive, so a zoom level of exactly 320,000,000 would *not* apply here.) Aside from the scale, the only difference between this rule and the first is the size of the symbol, which is set to 8 pixels on **line 18**.

The third rule (**lines 23-31**) is the largest scale denominator, corresponding to when the map is "zoomed out". The scale rule is set on **line 24**, so that the rule will apply to any map with a scale denominator of 320,000,000 or more. Again, the only other difference between this rule and the others is the size of the symbol, which is set to 4 pixels on **line 27**.

The result of this style is that points are drawn larger as one zooms in and smaller as one zooms out.

Lines

While lines can also seem to be simple shapes, having length but no width, there are many options and tricks for making lines display nicely.

Example lines layer

The `lines layer` used in the examples below contains road information for a fictional country. For reference, the attribute table for the points in this layer is included below.

`fid` (Feature ID)	`name` (Road name)	`type` (Road class)
line.1	Latway	highway
line.2	Crescent Avenue	secondary
line.3	Forest Avenue	secondary
line.4	Longway	highway
line.5	Saxer Avenue	secondary
line.6	Ridge Avenue	secondary
line.7	Holly Lane	local-road
line.8	Mulberry Street	local-road
line.9	Nathan Lane	local-road
line.10	Central Street	local-road
line.11	Lois Lane	local-road
line.12	Rocky Road	local-road
line.13	Fleet Street	local-road
line.14	Diane Court	local-road
line.15	Cedar Trail	local-road
line.16	Victory Road	local-road
line.17	Highland Road	local-road
line.18	Easy Street	local-road
line.19	Hill Street	local-road
line.20	Country Road	local-road
line.21	Main Street	local-road
line.22	Jani Lane	local-road
line.23	Shinbone Alley	local-road
line.24	State Street	local-road
line.25	River Road	local-road

Download the lines shapefile

Simple line

This example specifies lines be colored black with a thickness of 3 pixels.

Code

Download the "Simple line" YSLD

```
title: 'YSLD Cook Book: Simple Line'
feature-styles:
- name: name
  rules:
  - symbolizers:
    - line:
```

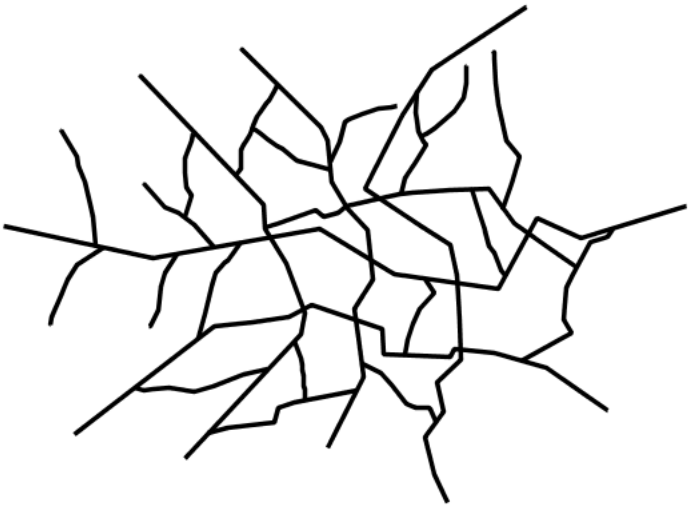


Fig. 6.233: Simple line

```
7                                   stroke-color: '#000000'
8                                   stroke-width: 3
```

Details

There is one rule in one feature style for this YSLD, which is the simplest possible situation. (All subsequent examples will contain one rule and one feature style unless otherwise specified.) Styling lines is accomplished via the line symbolizer (**lines 5-8**). **Line 7** specifies the color of the line to be black (`'#000000'`), while **line 8** specifies the width of the lines to be 3 pixels.

Line with border

This example shows how to draw lines with borders (sometimes called "cased lines"). In this case the lines are drawn with a 3 pixel blue center and a 1 pixel wide gray border.

Code

Download the "Line with border" YSLD

```
1                                   title: 'YSLD Cook Book: Line with border'
2                                   feature-styles:
3                                   - name: name
4                                     rules:
5                                     - symbolizers:
6                                       - line:
```

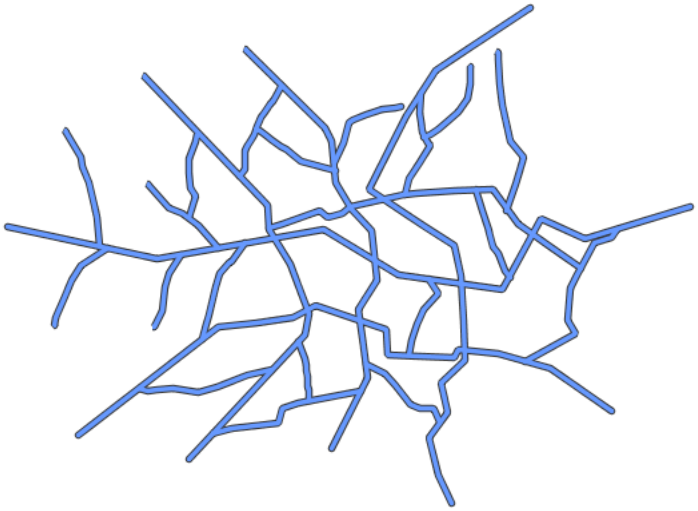


Fig. 6.234: Line with border

```
7                                   stroke-color: '#333333'
8                                   stroke-width: 5
9                                   stroke-linecap: round
10                              - name: name
11                                rules:
12                                - symbolizers:
13                                  - line:
14                                      stroke-color: '#6699FF'
15                                      stroke-width: 3
16                                      stroke-linecap: round
```

Details

Lines in YSLD have no notion of a "fill", only "stroke". Thus, unlike points or polygons, it is not possible to style the "edge" of the line geometry. It is, however, possible to achieve this effect by drawing each line twice: once with a certain width and again with a slightly smaller width. This gives the illusion of fill and stroke by obscuring the larger lines everywhere except along the edges of the smaller lines.

Since every line is drawn twice, the order of the rendering is *very* important. GeoServer renders `feature-styles` in the order that they are presented in the YSLD. In this style, the gray border lines are drawn first via the first feature style, followed by the blue center lines in a second feature style. This ensures that the blue lines are not obscured by the gray lines, and also ensures proper rendering at intersections, so that the blue lines "connect".

In this example, **lines 3-9** comprise the first feature style, which is the outer line (or "stroke"). **Line 7** specifies the color of the line to be

dark gray (`'#333333'`), **line 8** specifies the width of this line to be 5 pixels, and in **line 9** a `stroke-linecap` parameter of `round` renders the ends of the line as rounded instead of flat. (When working with bordered lines using a round line cap ensures that the border connects properly at the ends of the lines.)

Lines 10-16 comprise the second `feature-style`, which is the the inner line (or "fill"). **Line 14** specifies the color of the line to be a medium blue (`'#6699FF'`), **line 15** specifies the width of this line to be 3 pixels, and **line 16** again renders the edges of the line to be rounded instead of flat.

The result is a 3 pixel blue line with a 1 pixel gray border, since the 5 pixel gray line will display 1 pixel on each side of the 3 pixel blue line.

Dashed line

This example alters the *Simple line* to create a dashed line consisting of 5 pixels of drawn line alternating with 2 pixels of blank space.

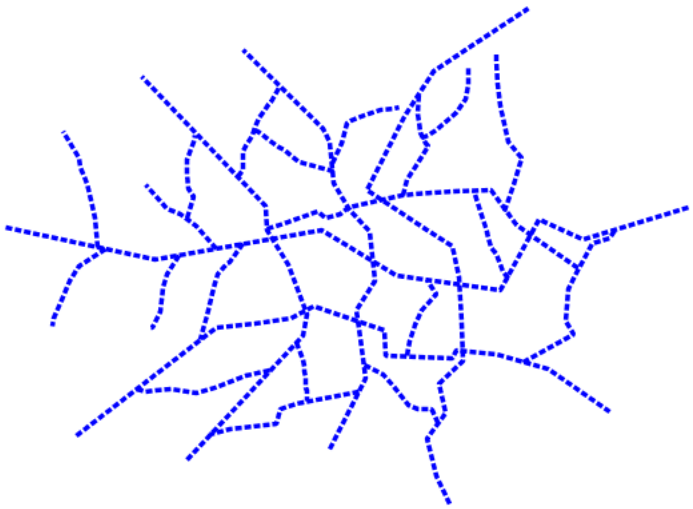


Fig. 6.235: Dashed line

Code

Download the "Dashed line" YSLD

```
title: 'YSLD Cook Book: Dashed line'
feature-styles:
- name: name
  rules:
  - symbolizers:
    - line:
```

```
7                                       stroke-color: '#0000FF'
8                                       stroke-width: 3
9                                       stroke-dasharray: 5 2
```

Details

In this example, **line 8** sets the color of the lines to be blue (`'#0000FF'`) and **line 8** sets the width of the lines to be 3 pixels. **Line 9** determines the composition of the line dashes. The value of `5 2` creates a repeating pattern of 5 pixels of drawn line, followed by 2 pixels of omitted line.

Offset line

This example alters the *Simple line* to add a perpendicular offset line on the left side of the line, at five pixels distance.

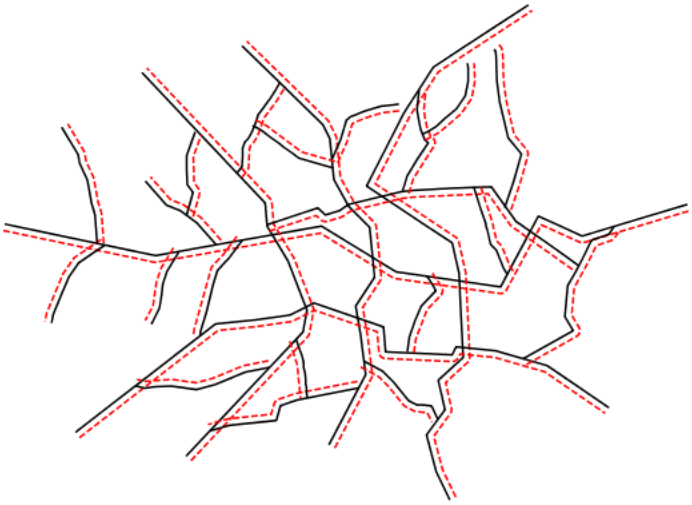


Fig. 6.236: Dashed line

Code

Download the "Offset line" YSLD

```
1                                   title: 'YSLD Cook Book: Dashed line'
2                                   feature-styles:
3                                   - name: name
4                                     rules:
5                                     - symbolizers:
6                                       - line:
7                                           stroke-color: '#000000'
```

```
8                                       stroke-width: 2
9                                   - line:
10                                      stroke-color: '#0000FF'
11                                      stroke-width: 3
12                                      stroke-dasharray: 5 2
13                                      offset: 3
```

Details

In this example, **lines 6-8** draw a simple black line like in the Simple line example. **Lines 9-12** draw a blue dashed line like in the above Dashed line example. **Line 13** modifies the dashed line with a 3 pixel offset from the line geometry.

Railroad (hatching)

This example uses hatching to create a railroad style. Both the line and the hatches are black, with a 2 pixel thickness for the main line and a 1 pixel width for the perpendicular hatches.

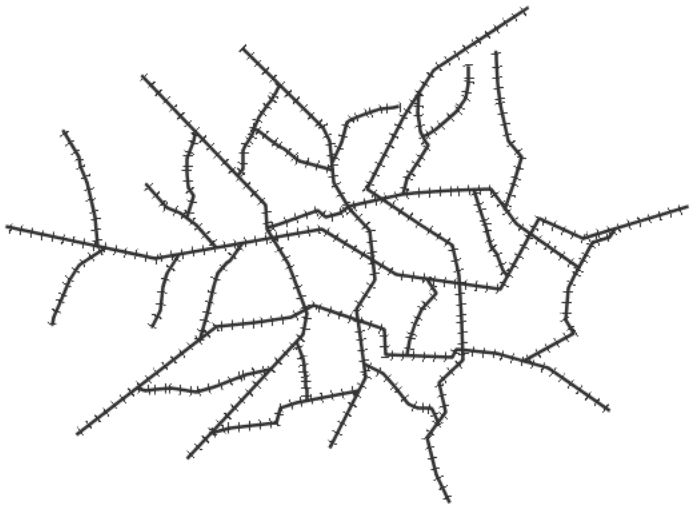


Fig. 6.237: Railroad (hatching)

Code

Download the "Railroad (hatching)" YSLD

```
1                                   title: 'YSLD Cook Book: Railroad (hatching)'
2                                   feature-styles:
3                                   - name: name
4                                     rules:
```

```
5                                   - symbolizers:
6                                     - line:
7                                         stroke-color: '#333333'
8                                         stroke-width: 3
9                                     - line:
10                                        stroke-color: '#333333'
11                                        stroke-width: 1
12                                        stroke-graphic-stroke:
13                                          size: 12
14                                          symbols:
15                                          - mark:
16                                              shape: shape://vertline
17                                              stroke-color: '#333333'
18                                              stroke-width: 1
```

Details

In this example there are two line symbolizers. The first symbolizer, on **lines 6-8**, draws a standard line, with **line 7** drawing the lines as dark gray (`'#333333'`) and **line 8** setting the width of the lines to be 2 pixels.

The hatching is invoked in the second symbolizer, on **lines 9-18**. **Line 16** specifies that the symbolizer draw a vertical line hatch (`shape://vertline`) perpendicular to the line geometry. **Lines 17-18** set the hatch color to dark gray (`'#333333'`) and width to 1 pixel. Finally, **line 13** specifies both the length of the hatch and the distance between each hatch to both be 12 pixels.

Spaced graphic symbols

This example uses a graphic stroke along with dash arrays to create a "dot and space" line type. Adding the dash array specification allows to control the amount of space between one symbol and the next one. Without using the dash array the lines would be densely populated with dots, each one touching the previous one.

Code

`Download the "Spaced symbols" YSLD`

```
1                                   name: Default Styler
2                                   title: 'YSLD Cook Book: Dash/Space line'
3                                   feature-styles:
4                                   - name: name
5                                     rules:
6                                     - symbolizers:
7                                       - line:
8                                           stroke-color: '#333333'
9                                           stroke-width: 1
10                                          stroke-dasharray: 4 6
11                                          stroke-graphic-stroke:
12                                            size: 4
```

Fig. 6.238: Spaced symbols along a line

```
13                                          symbols:
14                                          - mark:
15                                              shape: circle
16                                              stroke-color: '#333333'
17                                              stroke-width: 1
18                                              fill-color: '#666666'
```

Details

This example, like others before, uses a `stroke-graphic-stroke` to place a graphic symbol along a line. The symbol, defined on **lines 14-18** is a 4 pixel gray circle with a dark gray outline. The spacing between symbols is controlled with the `stroke-dasharray` at **line 9**, which specifies 4 pixels of pen-down (just enough to draw the circle) and 6 pixels of pen-up, to provide the spacing.

Alternating symbols with dash offsets

This example shows how to create a complex line style which alternates a dashed line and a graphic symbol. The code builds on features shown in the previous examples:

- `stroke-dasharray` controls pen-down/pen-up behavior to generate dashed lines
- `stroke-graphic-stroke` places symbols along a line
- combining the two allows control of symbol spacing

This also shows the usage of a *dash offset*, which controls where rendering starts in the dash array. For example, with a dash array of `5 10` and a dash offset of `7` the renderer starts drawing the pattern 7 pixels from the beginning. It skips the 5 pixels pen-down section and 2 pixels of the pen-up section, then draws the remaining 8 pixels of pen-up, then 5 down, 10 up, and so on.

The example shows how to use these features to create two synchronized sequences of dash arrays, one drawing line segments and the other symbols.

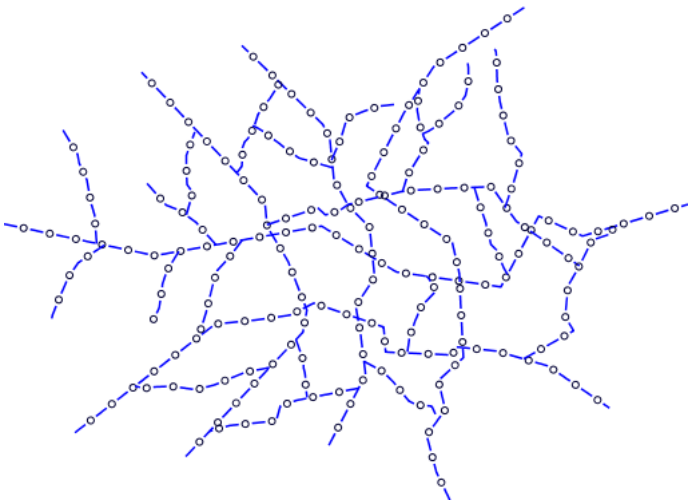


Fig. 6.239: Alternating dash and symbol

Code

Download the "Spaced symbols" YSLD

```
title: 'YSLD Cook Book: Dash/Symbol line'
feature-styles:
- name: name
  rules:
  - symbolizers:
    - line:
        stroke-color: '#0000FF'
        stroke-width: 1
        stroke-dasharray: 10 10
    - line:
        stroke-color: '#000033'
        stroke-width: 1
        stroke-dasharray: 5 15
        stroke-dashoffset: 7.5
        stroke-graphic-stroke:
          size: 5
          symbols:
          - mark:
              shape: circle
              stroke-color: '#000033'
              stroke-width: 1
```

Details

In this example two line symbolizers use `stroke-dasharray` and different symbology to produce a sequence of alternating dashes

and symbols. The first symbolizer (**lines 6-9**) is a simple dashed line alternating 10 pixels of pen-down with 10 pixels of pen-up. The second symbolizer (**lines 10-21**) alternates a 5 pixel empty circle with 15 pixels of white space. The circle symbol is produced by a `mark` element, with its symbology specified by `stroke` parameters (**lines 20-21**). The spacing between symbols is controlled with the `stroke-dasharray` (**line 13**), which specifies 5 pixels of pen-down (just enough to draw the circle) and 15 pixels of pen-up. In order to have the two sequences positioned correctly the second one uses a `stroke-dashoffset` of 7.5 (**line 14**). This makes the sequence start with 12.5 pixels of white space, then a circle (which is then centered between the two line segments of the other pattern), then 15 pixels of white space, and so on.

Line with default label

This example shows a text label on the simple line. This is how a label will be displayed in the absence of any other customization.

Fig. 6.240: Line with default label

Code

Download the "Line with default label" YSLD

```
1                                   name: Default Styler
2                                   title: 'YSLD Cook Book: Line with default label'
3                                   feature-styles:
4                                   - name: name
5                                     rules:
6                                     - symbolizers:
7                                       - line:
8                                           stroke-color: '#FF0000'
```

```
9                                       stroke-width: 1
10                                  - text:
11                                      label: ${name}
12                                      fill-color: '#000000'
13                                      font-family: Serif
14                                      font-size: 10
15                                      font-style: normal
16                                      font-weight: normal
17                                      placement: point
```

Details

In this example, there is one rule with a line symbolizer and a text symbolizer. The line symbolizer (**lines 6-8**) draws red lines (`'#FF0000'`). The text symbolizer (**lines 10-17**) determines the labeling of the lines. **Line 10** specifies that the text of the label will be determined by the value of the "name" attribute for each line. (Refer to the attribute table in the *Example lines layer* section if necessary.) **Line 11** sets the text color to black. All other details about the label are set to the renderer default, which here is Times New Roman font, font color black, and font size of 10 pixels.

Label following line

This example renders the text label to follow the contour of the lines.

Fig. 6.241: Label following line

Code

Download the "Label following line" YSLD

```
title: 'YSLD Cook Book: Label following line'
feature-styles:
- name: name
  rules:
  - symbolizers:
    - line:
        stroke-color: '#FF0000'
        stroke-width: 1
    - text:
        label: ${name}
        fill-color: '#000000'
        placement: line
        offset: 0
        x-followLine: true
```

Details

As the *Alternating symbols with dash offsets* example showed, the default label behavior isn't optimal. The label is displayed at a tangent to the line itself, leading to uncertainty as to which label corresponds to which line.

This example is similar to the *Alternating symbols with dash offsets* example with the exception of **lines 12-14**. **Line 14** sets the option to have the label follow the line, while **lines 12-13** specify that the label is placed along a line. If `placement: line` is not specified in an YSLD, then `placement: point` is assumed, which isn't compatible with line-specific rendering options.

Note: Not all labels are shown due to label conflict resolution. See the next section on *Optimized label placement* for an example of how to maximize label display.

Optimized label placement

This example optimizes label placement for lines such that the maximum number of labels are displayed.

Code

Download the "Optimized label" YSLD

```
title: 'YSLD Cook Book: Optimized label placement'
feature-styles:
- name: name
  rules:
  - symbolizers:
    - line:
```

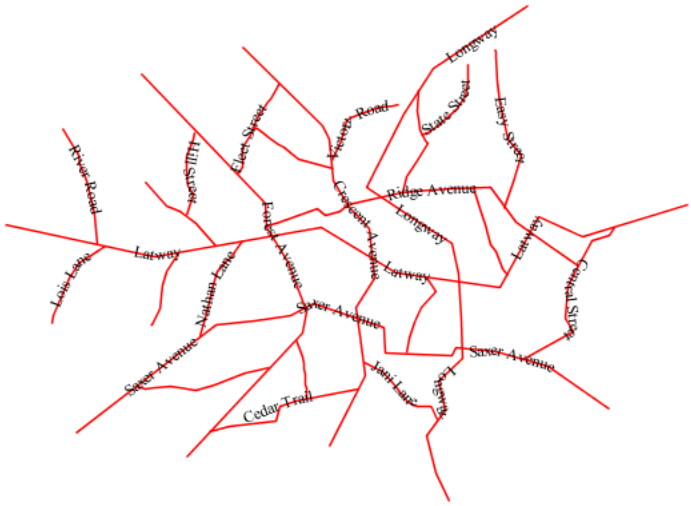


Fig. 6.242: Optimized label

```
7                                       stroke-color: '#FF0000'
8                                       stroke-width: 1
9                                   - text:
10                                      label: ${name}
11                                      fill-color: '#000000'
12                                      placement: line
13                                      offset: 0
14                                      x-followLine: true
15                                      x-maxAngleDelta: 90
16                                      x-maxDisplacement: 400
17                                      x-repeat: 150
```

Details

GeoServer uses "conflict resolution" to ensure that labels aren't drawn on top of other labels, obscuring them both. This accounts for the reason why many lines don't have labels in the previous example, *Label following line*. While this setting can be toggled, it is usually a good idea to leave it on and use other label placement options to ensure that labels are drawn as often as desired and in the correct places. This example does just that.

This example is similar to the previous example, *Label following line*. The only differences are contained in **lines 15-17**. **Line 15** sets the maximum angle that the label will follow. This sets the label to never bend more than 90 degrees to prevent the label from becoming illegible due to a pronounced curve or angle. **Line 16** sets the maximum displacement of the label to be 400 pixels. In order to resolve conflicts with overlapping labels, GeoServer will attempt to move the labels such that they are no longer overlapping. This value sets how far the label can be moved relative to its original placement. Finally,

line 17 sets the labels to be repeated every 150 pixels. A feature will typically receive only one label, but this can cause confusion for long lines. Setting the label to repeat ensures that the line is always labeled locally.

Optimized and styled label

This example improves the style of the labels from the *Optimized label placement* example.

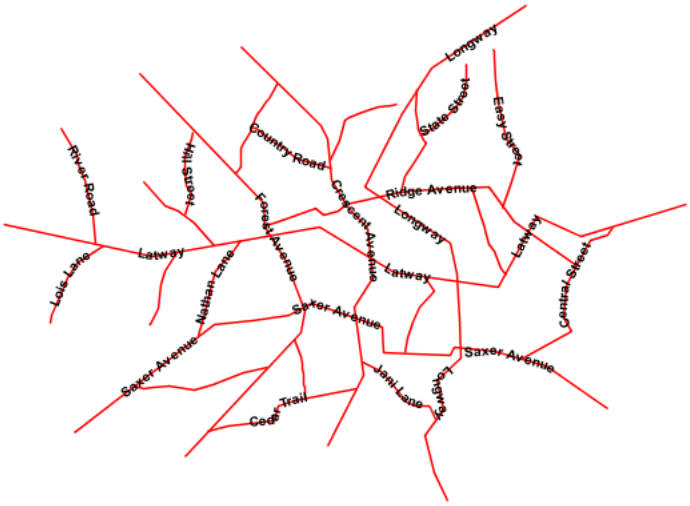


Fig. 6.243: Optimized and styled label

Code

Download the "Optimized and styled label" YSLD

```
title: 'YSLD Cook Book: Optimized and styled label'
feature-styles:
- name: name
  rules:
  - symbolizers:
    - line:
        stroke-color: '#FF0000'
        stroke-width: 1
    - text:
        label: ${name}
        fill-color: '#000000'
        font-family: Arial
        font-size: 10
        font-style: normal
        font-weight: bold
        placement: line
        offset: 0
```

```
                                    x-followLine: true
                                    x-maxAngleDelta: 90
                                    x-maxDisplacement: 400
                                    x-repeat: 150
```

Details

This example is similar to the *Optimized label placement*. The only difference is in the font information, which is contained in **lines 12-15**. **Line 12** sets the font family to be "Arial", **line 13** sets the font size to 10, **line 14** sets the font style to "normal" (as opposed to "italic" or "oblique"), and **line 15** sets the font weight to "bold" (as opposed to "normal").

Attribute-based line

This example styles the lines differently based on the "type" (Road class) attribute.

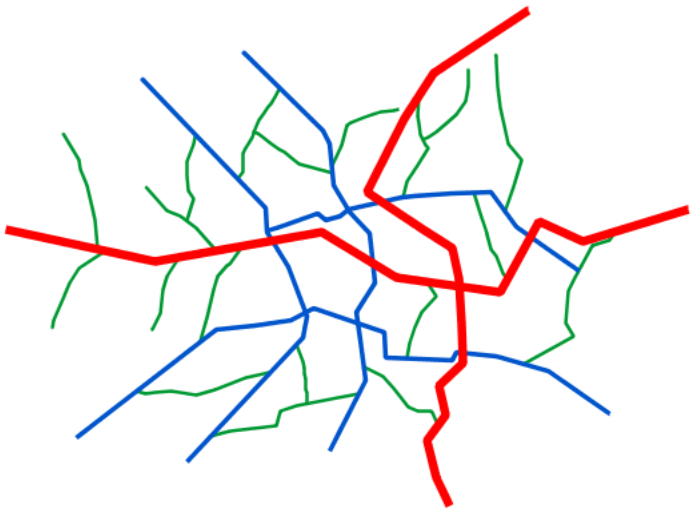


Fig. 6.244: Attribute-based line

Code

Download the "Attribute-based line" YSLD

```
                                title: 'YSLD Cook Book: Attribute-based line'
                                feature-styles:
                                - name: name
                                  rules:
                                  - name: local-road
```

```
6                                   filter: ${type = 'local-road'}
7                                   symbolizers:
8                                   - line:
9                                       stroke-color: '#009933'
10                                      stroke-width: 2
11                              - name: name
12                                rules:
13                                - name: secondary
14                                  filter: ${type = 'secondary'}
15                                  symbolizers:
16                                  - line:
17                                      stroke-color: '#0055CC'
18                                      stroke-width: 3
19                              - name: name
20                                rules:
21                                - name: highway
22                                  filter: ${type = 'highway'}
23                                  symbolizers:
24                                  - line:
25                                      stroke-color: '#FF0000'
26                                      stroke-width: 6
```

Details

Note: Refer to the *Example lines layer* to see the attributes for the layer. This example has eschewed labels in order to simplify the style, but you can refer to the example *Optimized and styled label* to see which attributes correspond to which points.

There are three types of road classes in our fictional country, ranging from back roads to high-speed freeways: "highway", "secondary", and "local-road". In order to handle each case separately, there is more than one feature style, each containing a single rule. This ensures that each road type is rendered in order, as each feature style is drawn based on the order in which it appears in the YSLD.

The three rules are designed as follows:

Rule order	Rule name / type	Color	Size
1	local-road	`#009933` (green)	2
2	secondary	`#0055CC` (blue)	3
3	highway	`#FF0000` (red)	6

Lines 3-10 comprise the first rule. **Line 6** sets the filter for this rule, such that the "type" attribute has a value of "local-road". If this condition is true for a particular line, the rule is rendered according to the line symbolizer which is on **lines 8-10**. **Lines 9-10** set the color of the line to be a dark green (`'#009933'`) and the width to be 2 pixels.

Lines 11-18 comprise the second rule. **Line 14** sets the filter for this rule, such that the "type" attribute has a value of "secondary". If this condition is true for a particular line, the rule is rendered according to the line symbolizer which is on **lines 16-18**. **Lines 17-18** set the

color of the line to be a dark blue (`'#0055CC'`) and the width to be 3 pixels, making the lines slightly thicker than the "local-road" lines and also a different color.

Lines 19-26 comprise the third and final rule. **Line 22** sets the filter for this rule, such that the "type" attribute has a value of "primary". If this condition is true for a particular line, the rule is rendered according to the line symbolizer which is on **lines 24-26**. **Lines 25-26** set the color of the line to be a bright red (`'#FF0000'`) and the width to be 6 pixels, so that these lines are rendered on top of and thicker than the other two road classes. In this way, the "primary" roads are given priority in the map rendering.

Zoom-based line

This example alters the *Simple line* style at different zoom levels.

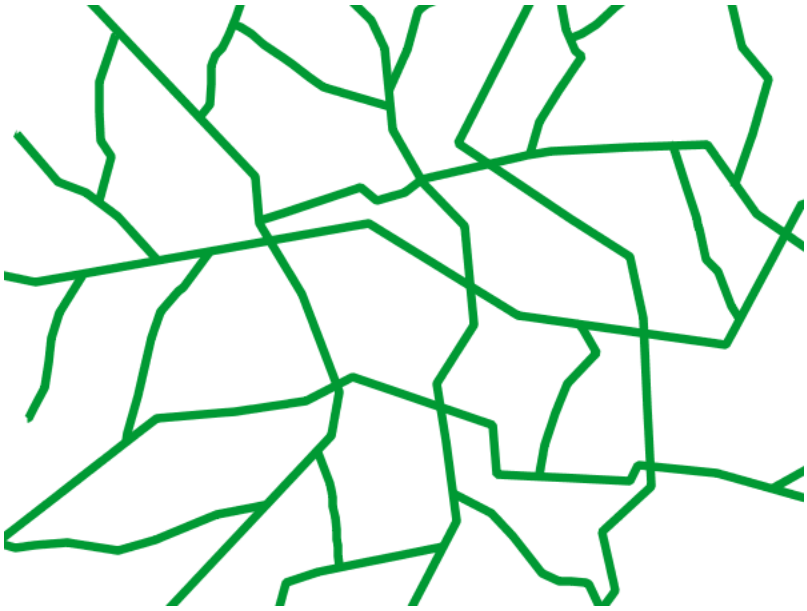


Fig. 6.245: Zoom-based line: Zoomed in

Code

Download the "Zoom-based line" YSLD

```
title: 'YSLD Cook Book: Zoom-based line'
feature-styles:
- name: name
  rules:
  - name: Large
    scale: [min,1.8e8]
    symbolizers:
    - line:
        stroke-color: '#009933'
```

Fig. 6.246: Zoom-based line: Partially zoomed

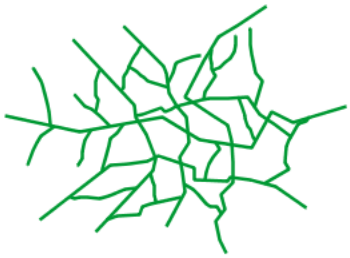


Fig. 6.247: Zoom-based line: Zoomed out

```
10                                      stroke-width: 6
11                                  - name: Medium
12                                    scale: [1.8e8,3.6e8]
13                                    symbolizers:
14                                    - line:
15                                        stroke-color: '#009933'
16                                        stroke-width: 4
17                                  - name: Small
18                                    scale: [3.6e8,max]
19                                    symbolizers:
20                                    - line:
21                                        stroke-color: '#009933'
22                                        stroke-width: 2
```

Details

It is often desirable to make shapes larger at higher zoom levels when creating a natural-looking map. This example varies the thickness of the lines according to the zoom level (or more accurately, scale denominator). Scale denominators refer to the scale of the map. A scale denominator of 10,000 means the map has a scale of 1:10,000 in the units of the map projection.

Note: Determining the appropriate scale denominators (zoom levels) to use is beyond the scope of this example.

This style contains three rules. The three rules are designed as follows:

Rule order	Rule name	Scale denominator	Line width
1	Large	1:180,000,000 or less	6
2	Medium	1:180,000,000 to 1:360,000,000	4
3	Small	Greater than 1:360,000,000	2

The order of these rules does not matter since the scales denominated in each rule do not overlap.

The first rule (**lines 5-10**) is the smallest scale denominator, corresponding to when the view is "zoomed in". The scale rule is set on **line 6**, so that the rule will apply to any map with a scale denominator of 180,000,000 or less. **Lines 9-10** draw the line to be dark green (`'#009933'`) with a width of 6 pixels.

The second rule (**lines 11-16**) is the intermediate scale denominator, corresponding to when the view is "partially zoomed". **Lines 12** set the scale such that the rule will apply to any map with scale denominators between 180,000,000 and 360,000,000. (The lower bound is inclusive and the upper bound is exclusive, so a zoom level of exactly 360,000,000 would *not* apply here.) Aside from the scale, the only difference between this rule and the previous is the width of the lines, which is set to 4 pixels on **line 16**.

The third rule (**lines 17-22**) is the largest scale denominator, corresponding to when the map is "zoomed out". The scale rule is set on **line 18**, so that the rule will apply to any map with a scale denominator of 360,000,000 or greater. Again, the only other difference between this rule and the others is the width of the lines, which is set to 2 pixels on **line 22**.

The result of this style is that lines are drawn with larger widths as one zooms in and smaller widths as one zooms out.

Polygons

Polygons are two dimensional shapes that contain both an outer edge (or "stroke") and an inside (or "fill"). A polygon can be thought of as an irregularly-shaped point and is styled in similar ways to points.

Example polygons layer

The `polygons layer` used below contains county information for a fictional country. For reference, the attribute table for the polygons is included below.

`fid` (Feature ID)	`name` (County name)	`pop` (Population)
polygon.1	Irony County	412234
polygon.2	Tracker County	235421
polygon.3	Dracula County	135022
polygon.4	Poly County	1567879
polygon.5	Bearing County	201989
polygon.6	Monte Cristo County	152734
polygon.7	Massive County	67123
polygon.8	Rhombus County	198029

`Download the polygons shapefile`

Simple polygon

This example shows a polygon filled in blue.

Code

`Download the "Simple polygon" YSLD`

```
1                                   title: 'YSLD Cook Book: Simple polygon'
2                                   feature-styles:
3                                   - name: name
4                                     rules:
5                                     - symbolizers:
6                                       - polygon:
7                                           fill-color: '#000080'
```

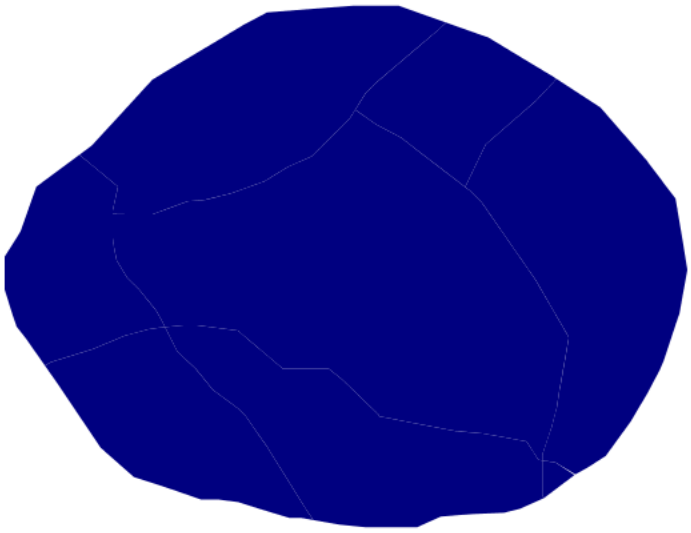


Fig. 6.248: Simple polygon

Details

There is one rule in one feature style for this style, which is the simplest possible situation. (All subsequent examples will share this characteristic unless otherwise specified.) Styling polygons is accomplished via the polygon symbolizer (**lines 6-7**). **Line 7** specifies dark blue (`'#000080'`) as the polygon's fill color.

Note: The light-colored borders around the polygons in the figure are artifacts of the renderer caused by the polygons being adjacent. There is no border in this style.

Simple polygon with stroke

This example adds a 2 pixel white stroke to the *Simple polygon* example.

Code

Download the "Simple polygon with stroke" YSLD

```
1                                   title: 'YSLD Cook Book: Simple polygon with stroke'
2                                   feature-styles:
3                                   - name: name
4                                     rules:
5                                     - symbolizers:
6                                       - polygon:
7                                           stroke-color: '#FFFFFF'
```

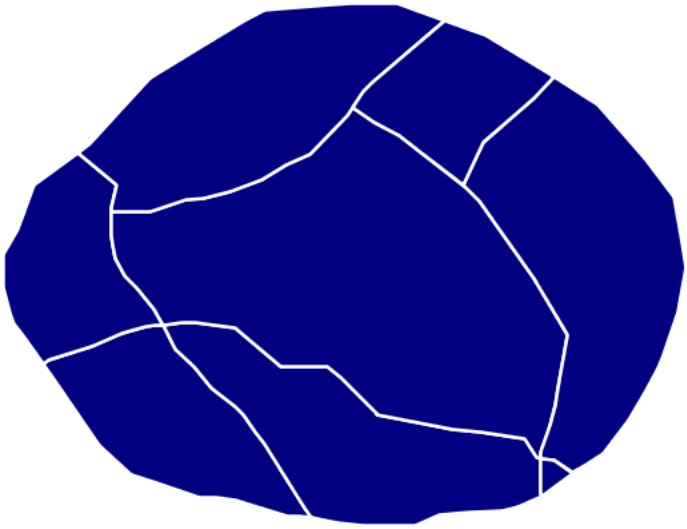


Fig. 6.249: Simple polygon with stroke

```
8                                       stroke-width: 2
9                                       fill-color: '#000080'
```

Details

This example is similar to the *Simple polygon* example above, with the addition of `stroke` parameters (**lines 7-8**). **Line 7** sets the color of stroke to white (`'#FFFFFF'`) and **line 8** sets the width of the stroke to 2 pixels.

Transparent polygon

This example builds on the *Simple polygon with stroke* example and makes the fill partially transparent by setting the opacity to 50%.

Code

Download the "Transparent polygon" YSLD

```
1                                   title: 'YSLD Cook Book: Transparent polygon'
2                                   feature-styles:
3                                   - name: name
4                                     rules:
5                                     - symbolizers:
6                                       - polygon:
7                                           stroke-color: '#FFFFFF'
8                                           stroke-width: 2
```

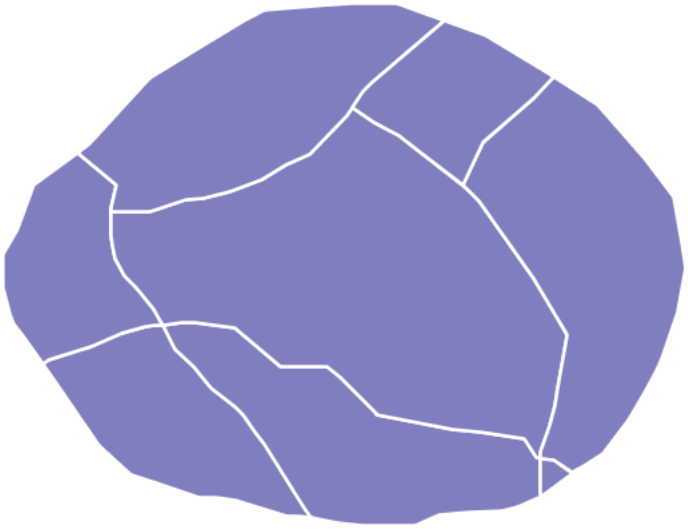


Fig. 6.250: Transparent polygon

```
9                                   fill-color: '#000080'
10                                  fill-opacity: 0.5
```

Details

This example is similar to the *Simple polygon with stroke* example, save for defining the fill's opacity in **line 10**. The value of 0.5 results in partially transparent fill that is 50% opaque. An opacity value of 1 would draw the fill as 100% opaque, while an opacity value of 0 would result in a completely transparent (0% opaque) fill. In this example, since the background is white, the dark blue looks lighter. Were the points imposed on a dark background, the resulting color would be darker.

Graphic fill

This example fills the polygons with a tiled graphic.

Code

Download the "Graphic fill" YSLD

```
1                                   title: 'YSLD Cook Book: Graphic fill'
2                                   feature-styles:
3                                   - name: name
4                                     rules:
5                                     - symbolizers:
6                                       - polygon:
```

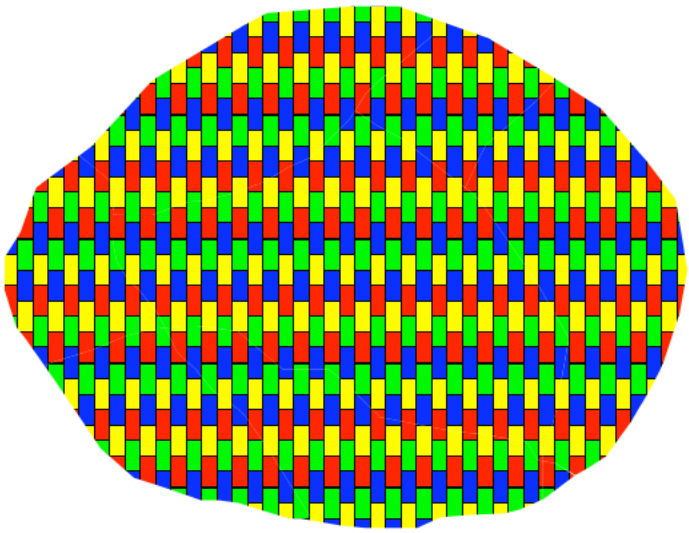


Fig. 6.251: Graphic fill

```
7                                   fill-color: '#808080'
8                                   fill-graphic:
9                                     size: 93
10                                    symbols:
11                                    - external:
12                                        url: colorblocks.png
13                                        format: image/png
```

Details

This style fills the polygon with a tiled graphic. This is known as an `external` in YSLD, to distinguish it from commonly-used shapes such as squares and circles that are "internal" to the renderer. **Lines 11-13** specify details for the graphic, with **line 12** setting the path and file name of the graphic and **line 13** indicating the file format (MIME type) of the graphic (`image/png`). Although a full URL could be specified if desired, no path information is necessary in **line 12** because this graphic is contained in the same directory as the YSLD. **Line 9** determines the height of the displayed graphic in pixels; if the value differs from the height of the graphic then it will be scaled accordingly while preserving the aspect ratio.

Fig. 6.252: Graphic used for fill

Hatching fill

This example fills the polygons with a hatching pattern.

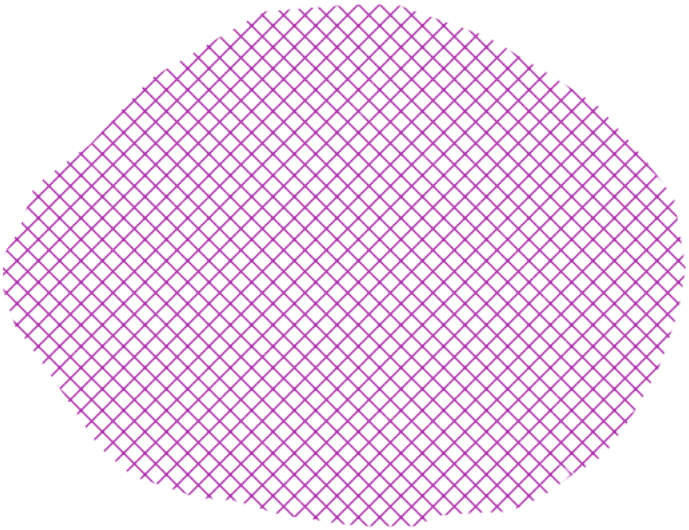


Fig. 6.253: Hatching fill

Code

Download the "Hatching fill" YSLD

```
1  title: 'YSLD Cook Book: Hatching fill'
2  feature-styles:
3  - name: name
4    rules:
5    - symbolizers:
6      - polygon:
7          fill-color: '#808080'
8          fill-graphic:
9            size: 16
10           symbols:
11           - mark:
12               shape: shape://times
13               stroke-color: '#990099'
14               stroke-width: 1
```

Details

In this example, there is a `fill-graphic` parameter as in the *Graphic fill* example, but a `mark` (**lines 11-14**) is used instead of an `external`. **Line 12** specifies a "times" symbol (an "x") be tiled throughout the polygon. **Line 13** sets the color to purple (`'#990099'`), **line 14** sets the width of the hatches to 1 pixel, and

line 9 sets the size of the tile to 16 pixels. Because hatch tiles are always square, the `size` sets both the width and the height.

Polygon with default label

This example shows a text label on the polygon. In the absence of any other customization, this is how a label will be displayed.

Fig. 6.254: Polygon with default label

Code

Download the "Polygon with default label" YSLD

```
1  title: 'YSLD Cook Book: Polygon with default label'
2  feature-styles:
3  - name: name
4    rules:
5    - symbolizers:
6      - polygon:
7          stroke-color: '#FFFFFF'
8          stroke-width: 2
9          fill-color: '#40FF40'
10     - text:
11         label: ${name}
12         placement: point
```

Details

In this example there is a polygon symbolizer and a text symbolizer. **Lines 6-9** comprise the polygon symbolizer. The fill of the polygon

is set on **line 7** to a light green (`'#40FF40'`) while the stroke of the polygon is set on **lines 8-9** to white (`'#FFFFFF'`) with a thickness of 2 pixels. The label is set in the text symbolizer on **lines 10-12**, with **line 11** determining what text to display, in this case the value of the "name" attribute. (Refer to the attribute table in the *Example polygons layer* section if necessary.) All other details about the label are set to the renderer default, which here is Times New Roman font, font color black, and font size of 10 pixels.

Label halo

This example alters the look of the *Polygon with default label* by adding a white halo to the label.

Fig. 6.255: Label halo

Code

Download the "Label halo" YSLD

```
1  title: 'YSLD Cook Book: Label halo'
2  feature-styles:
3  - name: name
4    rules:
5    - symbolizers:
6      - polygon:
7          stroke-color: '#FFFFFF'
8          stroke-width: 2
9          fill-color: '#40FF40'
10     - text:
11         label: ${name}
12         halo:
```

```
13                                    fill-color: '#FFFFFF'
14                                    radius: 3
15                                  placement:
16                                    type: point
```

Details

This example is similar to the *Polygon with default label*, with the addition of a halo around the labels on **lines 12-14**. A halo creates a color buffer around the label to improve label legibility. **Line 14** sets the radius of the halo, extending the halo 3 pixels around the edge of the label, and **line 13** sets the color of the halo to white (`'#FFFFFF'`). Since halos are most useful when set to a sharp contrast relative to the text color, this example uses a white halo around black text to ensure optimum readability.

Polygon with styled label

This example improves the label style from the *Polygon with default label* example by centering the label on the polygon, specifying a different font name and size, and setting additional label placement optimizations.

Fig. 6.256: Polygon with styled label

Code

Download the "Polygon with styled label" YSLD

```
title: 'YSLD Cook Book: Polygon with styled label'
feature-styles:
- name: name
  rules:
  - symbolizers:
    - polygon:
        stroke-color: '#FFFFFF'
        stroke-width: 2
        fill-color: '#40FF40'
    - text:
        label: ${name}
        fill-color: '#000000'
        font-family: Arial
        font-size: 11
        font-style: normal
        font-weight: bold
        placement: point
        anchor: [0.5,0.5]
        x-autoWrap: 60
        x-maxDisplacement: 150
```

Details

This example is similar to the *Polygon with default label* example, with additional styling options within the text symbolizer on lines **13-21**. **Lines 13-16** set the font styling. **Line 13** sets the font family to be Arial, **line 14** sets the font size to 11 pixels, **line 15** sets the font style to "normal" (as opposed to "italic" or "oblique"), and **line 16** sets the font weight to "bold" (as opposed to "normal").

The `anchor` parameter on **line 18** centers the label by positioning it 50% (or 0.5) of the way horizontally and vertically along the centroid of the polygon.

Finally, there are two added touches for label placement optimization: **line 20** ensures that long labels are split across multiple lines by setting line wrapping on the labels to 60 pixels, and **line 21** allows the label to be displaced by up to 150 pixels. This ensures that labels are compacted and less likely to spill over polygon boundaries. Notice little Massive County in the corner, whose label is now displayed."

Attribute-based polygon

This example styles the polygons differently based on the "pop" (Population) attribute.

Code

```
Download the "Attribute-based polygon" YSLD
```

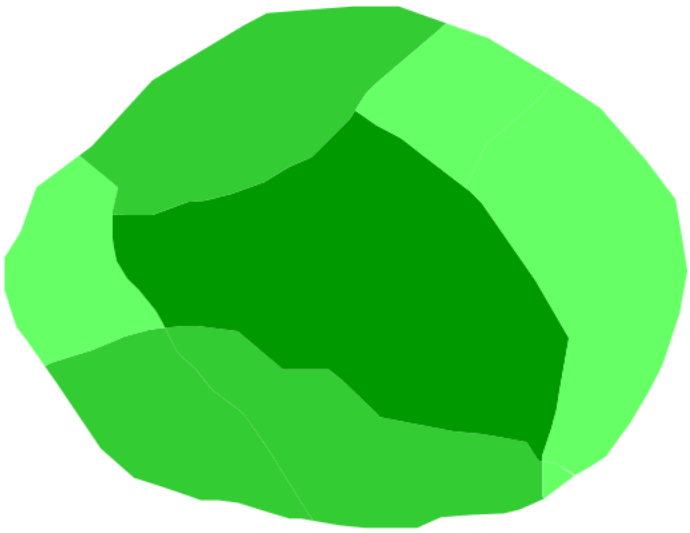


Fig. 6.257: Attribute-based polygon

```
title: 'YSLD Cook Book: Attribute-based polygon'
feature-styles:
- name: name
  rules:
  - name: SmallPop
    title: Less Than 200,000
    filter: ${pop < '200000'}
    symbolizers:
    - polygon:
        fill-color: '#66FF66'
  - name: MediumPop
    title: 200,000 to 500,000
    filter: ${pop >= '200000' AND pop < '500000'}
    symbolizers:
    - polygon:
        fill-color: '#33CC33'
  - name: LargePop
    title: ${Greater Than 500,000}
    filter: pop > '500000'
    symbolizers:
    - polygon:
        fill-color: '#009900'
```

Details

Note: Refer to the *Example polygons layer* to see the attributes for the layer. This example has eschewed labels in order to simplify the style, but you can refer to the example *Polygon with styled label* to see which attributes correspond to which polygons.

Each polygon in our fictional country has a population that is repre-

sented by the population ("pop") attribute. This style contains three rules that alter the fill based on the value of "pop" attribute, with smaller values yielding a lighter color and larger values yielding a darker color.

The three rules are designed as follows:

Rule order	Rule name	Population (`pop`)	Color
1	SmallPop	Less than 200,000	`#66FF66`
2	MediumPop	200,000 to 500,000	`#33CC33`
3	LargePop	Greater than 500,000	`#009900`

The order of the rules does not matter in this case, since each shape is only rendered by a single rule.

The first rule, on **lines 5-10**, specifies the styling of polygons whose population attribute is less than 200,000. **Line 7** sets this filter, denoting the attribute ("pop"), to be "less than" the value of 200,000. The color of the polygon fill is set to a light green (`'#66FF66'`) on **line 10**.

The second rule, on **lines 11-16**, is similar, specifying a style for polygons whose population attribute is greater than or equal to 200,000 but less than 500,000. The filter is set on **line 13**. This filter specifies two criteria instead of one: a "greater than or equal to" and a "less than" filter. These criteria are joined by `AND`, which mandates that both filters need to be true for the rule to be applicable. The color of the polygon fill is set to a medium green on (`'#33CC33'`) on **line 16**.

The third rule, on **lines 17-22**, specifies a style for polygons whose population attribute is greater than or equal to 500,000. The filter is set on **line 19**. The color of the polygon fill is the only other difference in this rule, which is set to a dark green (`'#009900'`) on **line 22**.

Zoom-based polygon

This example alters the style of the polygon at different zoom levels.

Code

Download the "Zoom-based polygon" YSLD

```
1                                   title: 'YSLD Cook Book: Zoom-based polygon'
2                                   feature-styles:
3                                   - name: name
4                                     rules:
5                                     - name: Large
6                                       scale: [min,1.0e8]
7                                       symbolizers:
8                                       - polygon:
9                                           stroke-color: '#000000'
```

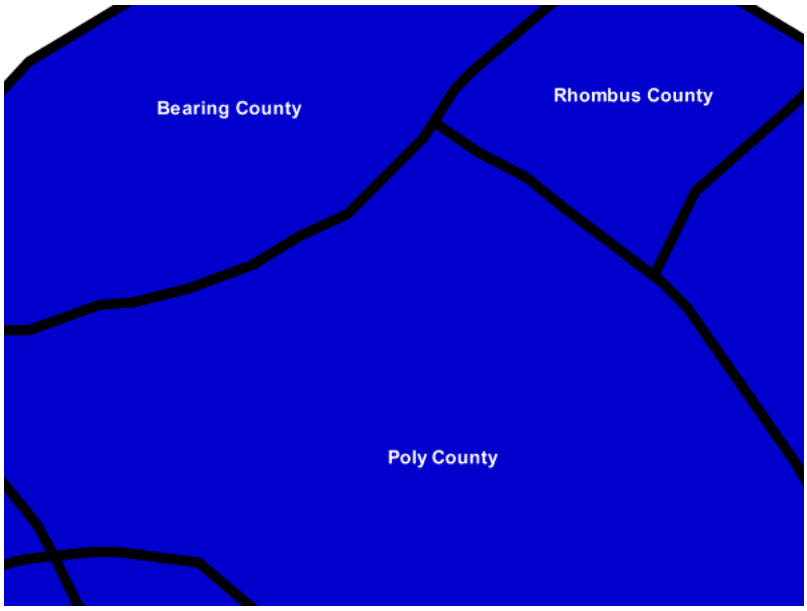


Fig. 6.258: Zoom-based polygon: Zoomed in

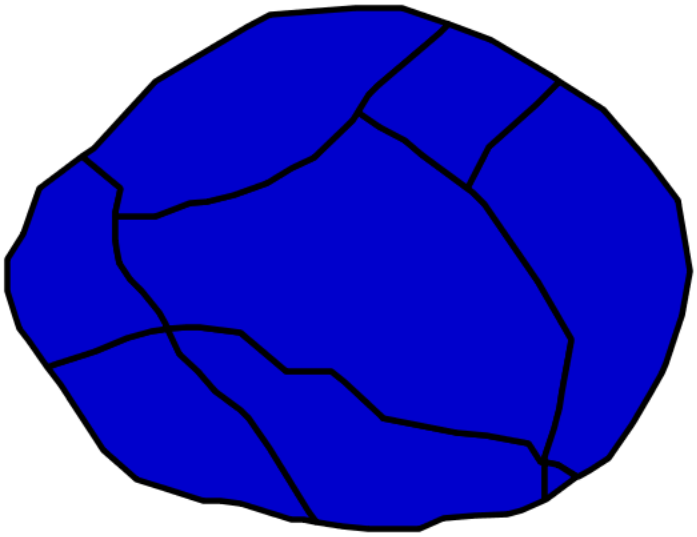


Fig. 6.259: Zoom-based polygon: Partially zoomed

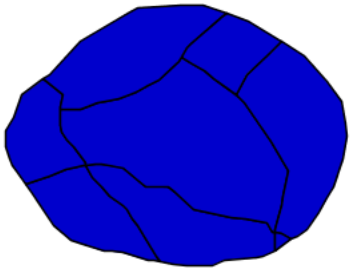


Fig. 6.260: Zoom-based polygon: Zoomed out

```
                stroke-width: 7
                fill-color: '#0000CC'
            - text:
                label: ${name}
                fill-color: '#FFFFFF'
                font-family: Arial
                font-size: 14
                font-style: normal
                font-weight: bold
                placement: point
                anchor: [0.5,0.5]
        - name: Medium
          scale: [1.0e8,2.0e8]
          symbolizers:
          - polygon:
                stroke-color: '#000000'
                stroke-width: 4
                fill-color: '#0000CC'
        - name: Small
          scale: [2.0e8,max]
          symbolizers:
          - polygon:
                stroke-color: '#000000'
                stroke-width: 1
                fill-color: '#0000CC'
```

Details

It is often desirable to make shapes larger at higher zoom levels when creating a natural-looking map. This example varies the thickness of the lines according to the zoom level. Polygons already do

this by nature of being two dimensional, but another way to adjust styling of polygons based on zoom level is to adjust the thickness of the stroke (to be larger as the map is zoomed in) or to limit labels to only certain zoom levels. This is ensures that the size and quantity of strokes and labels remains legible and doesn't overshadow the polygons themselves.

Zoom levels (or more accurately, scale denominators) refer to the scale of the map. A scale denominator of 10,000 means the map has a scale of 1:10,000 in the units of the map projection.

Note: Determining the appropriate scale denominators (zoom levels) to use is beyond the scope of this example.

This style contains three rules, defined as follows:

Rule order	Rule name	Scale denominator	Stroke width	Label display?
1	Large	1:100,000,000 or less	7	Yes
2	Medium	1:100,000,000 to 1:200,000,000	4	No
3	Small	Greater than 1:200,000,000	2	No

The first rule, on **lines 5-20**, is for the smallest scale denominator, corresponding to when the view is "zoomed in". The scale rule is set on **line 6** such that the rule will apply only where the scale denominator is 100,000,000 or less. **Line 11** defines the fill as blue (`'#0000CC'`). Note that the fill is kept constant across all rules regardless of the scale denominator. As in the *Polygon with default label* or *Polygon with styled label* examples, the rule also contains a text symbolizer at **lines 12-20** for drawing a text label on top of the polygon. **Lines 15-18** set the font information to be Arial, 14 pixels, and bold with no italics. The label is centered both horizontally and vertically along the centroid of the polygon on by setting `anchor` to be `[0.5, 0.5]` (or 50%) on **line 20**. Finally, the color of the font is set to white (`'#FFFFFF'`) in **line 14**.

The second rule, on **lines 21-27**, is for the intermediate scale denominators, corresponding to when the view is "partially zoomed". The scale rules on **lines 22** set the rule such that it will apply to any map with a scale denominator between 100,000,000 and 200,000,000. (The lower bound is inclusive and the upper bound is exclusive, so a zoom level of exactly 200,000,000 would *not* apply here.) Aside from the scale, there are two differences between this rule and the first: the width of the stroke is set to 4 pixels on **line 26** and a text symbolizer is not present so that no labels will be displayed.

The third rule, on **lines 28-34**, is for the largest scale denominator, corresponding to when the map is "zoomed out". The scale rule is set on **line 29** such that the rule will apply to any map with a scale denominator of 200,000,000 or greater. Again, the only differences between this rule and the others are the width of the lines, which is set to 1 pixel on **line 33**, and the absence of a text symbolizer so that no labels will be displayed.

The resulting style produces a polygon stroke that gets larger as one zooms in and labels that only display when zoomed in to a sufficient level.

Rasters

Rasters are geographic data displayed in a grid. They are similar to image files such as PNG files, except that instead of each point containing visual information, each point contains geographic information in numerical form. Rasters can be thought of as a georeferenced table of numerical values.

One example of a raster is a Digital Elevation Model (DEM) layer, which has elevation data encoded numerically at each georeferenced data point.

Example raster

The `raster layer` that is used in the examples below contains elevation data for a fictional world. The data is stored in EPSG:4326 (longitude/latitude) and has a data range from 70 to 256. If rendered in grayscale, where minimum values are colored black and maximum values are colored white, the raster would look like this:

Fig. 6.261: Raster file as rendered in grayscale

`Download the raster shapefile`

Two-color gradient

This example shows a two-color style with green at lower elevations and brown at higher elevations.

Code

`Download the "Two-color gradient" YSLD`

Fig. 6.262: Two-color gradient

```
1  title: 'YSLD Cook Book: Two color gradient'
2  feature-styles:
3  - name: name
4    rules:
5    - symbolizers:
6      - raster:
7          opacity: 1.0
8          color-map:
9            type: ramp
10           entries:
11           - ['#008000',1,70,'']
12           - ['#663333',1,256,'']
```

Details

There is one rule in one feature style for this example, which is the simplest possible situation. All subsequent examples will share this characteristic. Styling of rasters is done via the raster symbolizer (**lines 2-7**).

This example creates a smooth gradient between two colors corresponding to two elevation values. The gradient is created via the `color-map` on **lines 8-12**. Each entry in the `color-map` represents one entry or anchor in the gradient. **Line 11** sets the lower value of 70 and color to a dark green (`'#008000'`). **Line 12** sets the upper value of 256 and color to a dark brown (`'#663333'`). **Line 9** sets the type to `ramp`, which means that all data values in between these two quantities will be linearly interpolated: a value of 163 (the midpoint between 70 and 256) will be colored as the midpoint between the two colors (in this case approximately `'#335717'`, a muddy green).

Transparent gradient

This example creates the same two-color gradient as in the *Two-color gradient* as in the example above but makes the entire layer mostly transparent by setting a 30% opacity.

Fig. 6.263: Transparent gradient

Code

Download the "Transparent gradient" YSLD

```
title: 'YSLD Cook Book: Transparent gradient'
feature-styles:
- name: name
  rules:
  - symbolizers:
    - raster:
        opacity: 0.3
        color-map:
          type: ramp
          entries:
          - ['#008000',1,70,'']
          - ['#663333',1,256,'']
```

Details

This example is similar to the *Two-color gradient* example save for the addition of **line 7**, which sets the opacity of the layer to 0.3 (or 30% opaque). An opacity value of 1 means that the shape is drawn 100% opaque, while an opacity value of 0 means that the shape is rendered as completely transparent. The value of 0.3 means that the the raster partially takes on the color and style of whatever is drawn beneath it. Since the background is white in this example, the colors generated from the `color-map` look lighter, but were the raster imposed on a dark background the resulting colors would be darker.

Brightness and contrast

This example normalizes the color output and then increases the brightness by a factor of 2.

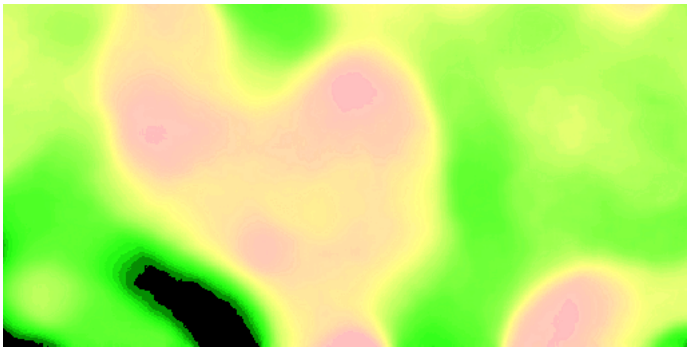


Fig. 6.264: Brightness and contrast

Code

Download the "Brightness and contrast" YSLD

```
1  title: 'YSLD Cook Book: Brightness and contrast'
2  feature-styles:
3  - name: name
4    rules:
5    - symbolizers:
6      - raster:
7          opacity: 1
8          color-map:
9            type: ramp
10           entries:
11           - ['#008000',1,70,'']
12           - ['#663333',1,256,'']
13         contrast-enhancement:
14           mode: normalize
15           gamma: 0.5
```

Details

This example is similar to the *Two-color gradient*, save for the addition of the `contrast-enhancement` parameter on **lines 13-15**. **Line 14** normalizes the output by increasing the contrast to its maximum extent. **Line 15** then adjusts the brightness by a factor of 0.5. Since values less than 1 make the output brighter, a value of 0.5 makes the output twice as bright.

As with previous examples, **lines 8-12** determine the `color-map`, with **line 11** setting the lower bound (70) to be colored dark green (`'#008000'`) and **line 12** setting the upper bound (256) to be colored dark brown (`'#663333'`).

Three-color gradient

This example creates a three-color gradient in primary colors.

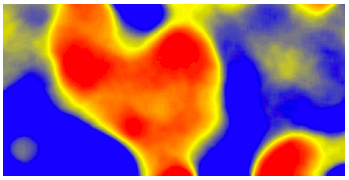


Fig. 6.265: Three-color gradient

Code

Download the "Three-color gradient" YSLD

```
title: 'YSLD Cook Book: Three color gradient'
feature-styles:
- name: name
  rules:
  - symbolizers:
    - raster:
        opacity: 1
        color-map:
          type: ramp
          entries:
          - ['#0000FF',1,150,'']
          - ['#FFFF00',1,200,'']
          - ['#FF0000',1,250,'']
```

Details

This example creates a three-color gradient based on a `color-map` with three entries on **lines 8-13**: **line 11** specifies the lower bound (150) be styled in blue (`'#0000FF'`), **line 12** specifies an intermediate point (200) be styled in yellow (`'#FFFF00'`), and **line 13** specifies the upper bound (250) be styled in red (`'#FF0000'`).

Since our data values run between 70 and 256, some data points are not accounted for in this style. Those values below the lowest entry in the color map (the range from 70 to 149) are styled the same color as the lower bound, in this case blue. Values above the upper bound in the color map (the range from 251 to 256) are styled the same color as the upper bound, in this case red.

Alpha channel

This example creates an "alpha channel" effect such that higher values are increasingly transparent.

Code

Download the "Alpha channel" YSLD

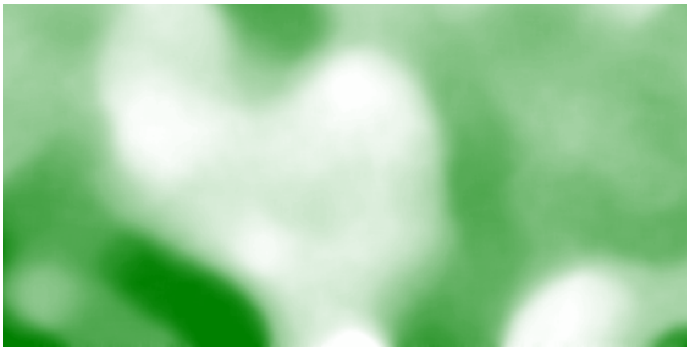


Fig. 6.266: Alpha channel

```
1  title: 'YSLD Cook Book: Alpha channel'
2  feature-styles:
3  - name: name
4    rules:
5    - symbolizers:
6      - raster:
7          opacity: 1
8          color-map:
9            type: ramp
10           entries:
11           - ['#008000',1,70,'']
12           - ['#008000',0,256,'']
```

Details

An alpha channel is another way of referring to variable transparency. Much like how a gradient maps values to colors, each entry in a `color-map` can have a value for opacity (with the default being 1.0 or completely opaque).

In this example, there is a `color-map` with two entries: **line 11** specifies the lower bound of 70 be colored dark green (`'#008000'`), while **line 13** specifies the upper bound of 256 also be colored dark green but with an opacity value of 0. This means that values of 256 will be rendered at 0% opacity (entirely transparent). Just like the gradient color, the opacity is also linearly interpolated such that a value of 163 (the midpoint between 70 and 256) is rendered at 50% opacity.

Discrete colors

This example shows a gradient that is not linearly interpolated but instead has values mapped precisely to one of three specific colors.

Code

Download the "Discrete colors" YSLD

Fig. 6.267: Discrete colors

```
1  title: 'YSLD Cook Book: Discrete colors'
2  feature-styles:
3  - name: name
4    rules:
5    - symbolizers:
6      - raster:
7          opacity: 1
8          color-map:
9            type: intervals
10           entries:
11           - ['#008000',1,150,'']
12           - ['#663333',1,256,'']
```

Details

Sometimes color bands in discrete steps are more appropriate than a color gradient. The `type: intervals` parameter added to the `color-map` on **line 9** sets the display to output discrete colors instead of a gradient. The values in each entry correspond to the upper bound for the color band such that colors are mapped to values less than the value of one entry but greater than or equal to the next lower entry. For example, **line 11** colors all values less than 150 to dark green (`'#008000'`) and **line 12** colors all values less than 256 but greater than or equal to 150 to dark brown (`'#663333'`).

Many color gradient

This example shows a gradient interpolated across eight different colors.

Code

Download the "Many color gradient" YSLD

```
1  title: 'YSLD Cook Book: Many color gradient'
2  feature-styles:
3  - name: name
```

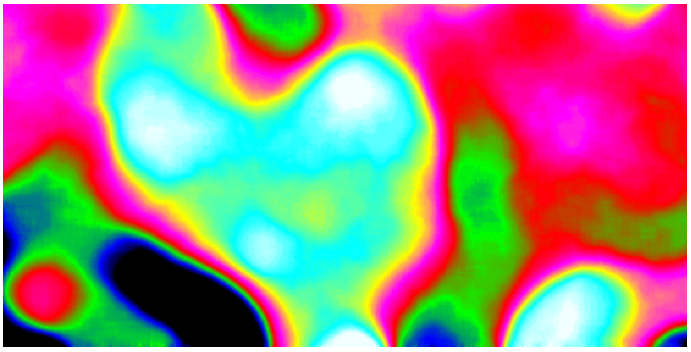


Fig. 6.268: Many color gradient

```
4                                   rules:
5                                   - symbolizers:
6                                     - raster:
7                                         opacity: 1
8                                         color-map:
9                                           type: ramp
10                                          entries:
11                                          - ['#000000',1,95,'']
12                                          - ['#0000FF',1,110,'']
13                                          - ['#00FF00',1,135,'']
14                                          - ['#FF0000',1,160,'']
15                                          - ['#FF00FF',1,185,'']
16                                          - ['#FFFF00',1,210,'']
17                                          - ['#00FFFF',1,235,'']
18                                          - ['#FFFFFF',1,256,'']
```

Details

A `color-map` can include up to 255 entries. This example has eight entries (**lines 11-18**):

Entry number	Value	Color	RGB code
1	95	Black	`'#000000'`
2	110	Blue	`'#0000FF'`
3	135	Green	`'#00FF00'`
4	160	Red	`'#FF0000'`
5	185	Purple	`'#FF00FF'`
6	210	Yellow	`'#FFFF00'`
7	235	Cyan	`'#00FFFF'`
8	256	White	`'#FFFFFF'`

6.6 MBStyle Styling

This module adds support for the MBStyle language.

MBStyle is a JSON based language which can be converted to SLD, the internal data model that GeoServer's renderer uses. For details on Mapbox Style syntax see the reference below.

MBStyle is not a part of GeoServer by default, but is available as an optional install.

6.6.1 Installing the GeoServer MBStyle extension

1. Download the extension from the nightly GeoServer community module builds.

> **Warning:** Make sure to match the version of the extension to the version of the GeoServer instance!

2. Extract the contents of the archive into the `WEB-INF/lib` directory of the GeoServer installation.

6.6.2 Publishing a GeoServer Layer for use with Mapbox Styles

GeoServer can be configured to serve layers as vector tiles which can be used as sources for Mapbox styles rendered by client-side applications such as OpenLayers.

1. *Enable CORS* in GeoServer.
2. Install the *Vector Tiles* extension.
3. Follow the *Vector tiles tutorial* to publish your layers in `application/x-protobuf;type=mapbox-vector` format (You only need to do the "Publish vector tiles in GeoWebCache" step).

Once these steps are complete, you will be able to use your GeoServer layers in any Mapbox-compatible client application that can access your GeoServer.

The source syntax to use these GeoServer layers in your MapBox Style is:

```
"<source-name>": {
  "type": "vector",
  "tiles": [
    "http://localhost:8080/geoserver/
↪gwc/service/wmts?REQUEST=GetTile&SERVICE=WMTS
        &VERSION=1.0.0&LAYER=
↪<workspace>:<layer>&STYLE=&TILEMATRIX=EPSG:900913:{z}
        &TILEMATRIXSET=EPSG:900913&
↪FORMAT=application/x-protobuf;type=mapbox-vector
        &TILECOL={x}&TILEROW={y}"
  ],
  "minZoom": 0,
  "maxZoom": 14
}
```

Note: `<workspace>` and `<layer>` should be replaced by the workspace and name of the layer in question. `{x}`, `{y}`, and `{z}` are placeholder values for the tile indices and should be preserved as written.

Note: `<source-name>` should be replaced by a source name of your choice. It will be used to refer to the source elsewhere in the Mapbox Style.

Note: If geoserver is not being served from `localhost:8080`, update the domain accordingly.

6.6.3 MBStyle reference

This section will detail the usage and syntax of the MBStyle language.

As MBstyle is heavily modeled on JSON, it may be useful to refer to the JSON-Schema documentation for basic syntax.

For an extended reference to these styles check out the Mapbox Style Specifications.

Mapbox Styles Module

The **gs-mbstyle** module is an unsupported module that provides a parser/encoder to convert between Mapbox Styles and GeoServer style objects. These docs are under active development, along with the module itself.

References:

- https://www.mapbox.com/mapbox-gl-style-spec

MapBox Types

copied from the MapBox Style Specification

Color

Colors are written as JSON strings in a variety of permitted formats: HTML-style hex values, rgb, rgba, hsl, and hsla. Predefined HTML colors names, like yellow and blue, are also permitted.

```
{
  "line-color": "#ff0",
  "line-color": "#ffff00",
  "line-color": "rgb(255, 255, 0)",
  "line-color": "rgba(255, 255, 0, 1)",
  "line-color": "hsl(100, 50%, 50%)",
  "line-color": "hsla(100, 50%, 50%, 1)",
  "line-color": "yellow"
}
```

Especially of note is the support for hsl, which can be easier to reason about than rgb().

Enum

One of a fixed list of string values. Use quotes around values.

```
{
  "text-transform": "uppercase"
}
```

String

A string is basically just text. In Mapbox styles, you're going to put it in quotes. Strings can be anything, though pay attention to the case of text-field - it actually will refer to features, which you refer to by putting them in curly braces, as seen in the example below.

```
{
  "text-field": "{MY_FIELD}"
}
```

Boolean

Boolean means yes or no, so it accepts the values true or false.

```
{
  "fill-enabled": true
}
```

Number

A number value, often an integer or floating point (decimal number). Written without quotes.

```
{
  "text-size": 24
}
```

Array

Arrays are comma-separated lists of one or more numbers in a specific order. For example, they're used in line dash arrays, in which the numbers specify intervals of line, break, and line again.

```
{
  "line-dasharray": [2, 4]
}
```

Expressions

The value for any layout property, paint property, or filter may be specified as an expression. An expression defines a formula for computing the value of the property using the operators described below. The set of expression operators provided by Mapbox GL includes:

- Mathematical operators for performing arithmetic and other operations on numeric values
- Logical operators for manipulating boolean values and making conditional decisions
- String operators for manipulating strings
- Data operators, providing access to the properties of source features
- Camera operators, providing access to the parameters defining the current map view

Expressions are represented as JSON arrays. The first element of an expression array is a string naming the expression operator, e.g. `"*"` or `"case"`. Subsequent elements (if any) are the arguments to the expression. Each argument is either a literal value (a string, number, boolean, or null), or another expression array.

[expression_name, argument_0, argument_1, ...]

Data expressions

A *data expression* is any expression that access feature data – that is, any expression that uses one of the data operators: `get`, `has`, `id`, `geometry-type`, `properties`, or `feature-state`. Data expressions allow a feature's properties or state to determine its appearance. They can be used to differentiate features within the same layer and to create data visualizations.

```
{
    "circle-color": [
        "rgb",
        // red is␣
↪higher when feature.properties.temperature is higher
        ["get", "temperature"],
        // green is always zero
        0,
        // blue is␣
↪higher when feature.properties.temperature is lower
        ["-", 100, ["get", "temperature"]]
    ]
}
```

This example uses the `get` operator to obtain the temperature value of each feature. That value is used to compute arguments to the `rgb` operator, defining a color in terms of its red, green, and blue components.

Data expressions are allowed as the value of the `filter` property, and as values for most paint and layout properties. However, some paint and layout properties do not yet support data expressions. The level of support is indicated by the "data-driven styling" row of the "SDK Support" table for each property. Data expressions with the `feature-state` operator are allowed only on paint properties.

Camera expressions

A *camera expression* is any expression that uses the `zoom operator`. Such expressions allow the the appearance of a layer to change with

the map's zoom level. Camera expressions can be used to create the appearance of depth and to control data density.

```
{
    "circle-radius": [
        "interpolate", ["linear"], ["zoom"],
        // zoom is 5 (or less) -> circle radius will be 1px
        5, 1,
        //
        zoom is 10 (or greater) -> circle radius will be 5px
        10, 5
    ]
}
```

This example uses the `interpolate` operator to define a linear relationship between zoom level and circle size using a set of input-output pairs. In this case, the expression indicates that the circle radius should be 1 pixel when the zoom level is 5 or below, and 5 pixels when the zoom is 10 or above. In between, the radius will be linearly interpolated between 1 and 5 pixels

Camera expressions are allowed anywhere an expression may be used. However, when a camera expression used as the value of a layout or paint property, it must be in one of the following forms:

```
[ "interpolate", interpolation, ["zoom"], ... ]
```

Or:

```
[ "step", ["zoom"], ... ]
```

Or:

```
[
    "let",
    ... variable bindings...,
    [ "interpolate", interpolation, ["zoom"], ... ]
]
```

Or:

```
[
    "let",
    ... variable bindings...,
    [ "step", ["zoom"], ... ]
]
```

That is, in layout or paint properties, `["zoom"]` may appear only as the input to an outer `interpolate` or `step` expression, or such an expression within a `let` expression.

There is an important difference between layout and paint properties in the timing of camera expression evaluation. Paint property camera expressions are re-evaluated whenever the zoom level changes, even fractionally. For example, a paint property camera expression will be re-evaluated continuously as the map moves between zoom levels 4.1 and 4.6. On the other hand, a layout property

camera expression is evaluated only at integer zoom levels. It will not be re-evaluated as the zoom changes from 4.1 to 4.6 – only if it goes above 5 or below 4.

Composition

A single expression may use a mix of data operators, camera operators, and other operators. Such composite expressions allows a layer's appearance to be determined by a combination of the zoom level and individual feature properties.

```
{
    "circle-radius": [
        "interpolate", ["linear"], ["zoom"],
        // when zoom is 0, set each feature's␣
↪circle radius to the value of its "rating" property
        0, ["get", "rating"],

        ␣
↪// when zoom is 10, set each feature's circle radius␣
↪to four times the value of its "rating" property
        10, ["*", 4, ["get", "rating"]]
    ]
}
```

An expression that uses both data and camera operators is considered both a data expression and a camera expression, and must adhere to the restrictions described above for both.

Type system

The input arguments to expressions, and their result values, use the same set of types as the rest of the style specification: boolean, string, number, color, and arrays of these types. Furthermore, expressions are type safe: each use of an expression has a known result type and required argument types, and the SDKs verify that the result type of an expression is appropriate for the context in which it is used. For example, the result type of an expression in the `filter` property must be boolean, and the arguments to the `+` operator must be numbers.

When working with feature data, the type of a feature property value is typically not known ahead of time by the SDK. In order to preserve type safety, when evaluating a data expression, the SDK will check that the property value is appropriate for the context. For example, if you use the expression `["get", "feature-color"]` for the `circle-color` property, the SDK will verify that the `feature-color` value of each feature is a string identifying a valid color. If this check fails, an error will be indicated in an SDK-specific way (typically a log message), and the default value for the property will be used instead.

In most cases, this verification will occur automatically wherever it is needed. However, in certain situations, the SDK may be unable to automatically determine the expected result type of a data expression from surrounding context. For example, it is not clear whether the expression `["<", ["get", "a"], ["get", "b"]]` is attempting to compare strings or numbers. In situations like this, you can use one of the type assertion expression operators to in-

dicate the expected type of a data expression: `["<", ["number", ["get", "a"]], ["number", ["get", "b"]]]`. A type assertion checks that the feature data actually matches the expected type of the data expression. If this check fails, it produces an error and causes the whole expression to fall back to the default value for the property being defined. The assertion operators are `array`, `boolean`, `number`, and `string`.

Expressions perform only one kind of implicit type conversion: a data expression used in a context where a color is expected will convert a string representation of a color to a color value. In all other cases, if you want to convert between types, you must use one of the type conversion expression operators: `to-boolean`, `to-number`, `to-string`, or `to-color`. For example, if you have a feature property that stores numeric values in string format, and you want to use those values as numbers rather than strings, you can use an expression such as `["to-number", ["get", "property-name"]]`.

Function

Note: As of GeoTools 20.0 / MapBox 0.41.0, functions are deprecated. Use expressions instead.

The value for any layout or paint property may be specified as a function. Functions allow you to make the appearance of a map feature change with the current zoom level and/or the feature's properties.

stops

Required (except for identity functions) array.

Functions are defined in terms of input and output values. A set of one input value and one output value is known as a "stop."

property

Optional string

If specified, the function will take the specified feature property as an input. See *Zoom Functions and Property Functions* for more information.

base

Optional number. Default is 1.

The exponential base of the interpolation curve. It controls the rate at which the function output increases. Higher values make the output increase more towards the high end of the range. With values close to 1 the output increases linearly.

type

Optional enum. One of identity, exponential, interval, categorical

identity

functions return their input as their output.

exponential

functions generate an output by interpolating between stops just less than and just greater than the function input. The domain must be numeric.

interval

functions return the output value of the stop just less than the function input. The domain must be numeric.

categorical

functions return the output value of the stop equal to the function input.

default

A value to serve as a fallback function result when a value isn't otherwise available. It is used in the following circumstances:

- In categorical functions, when the feature value does not match any of the stop domain values.
- In property and zoom-and-property functions, when a feature does not contain a value for the specified property.
- In identity functions, when the feature value is not valid for the style property (for example, if the function is being used for a circle-color property but the feature property value is not a string or not a valid color).
- In interval or exponential property and zoom-and-property functions, when the feature value is not numeric.

If no default is provided, the style property's default is used in these circumstances.

colorSpace

Optional enum. One of rgb, lab, hcl

The color space in which colors interpolated. Interpolating colors in perceptual color spaces like LAB and HCL tend to produce color ramps that look more consistent and produce colors that can be differentiated more easily than those interpolated in RGB space.

rgb

Use the RGB color space to interpolate color values

lab

Use the LAB color space to interpolate color values.

hcl

Use the HCL color space to interpolate color values, interpolating the Hue, Chroma, and Luminance channels individually.

Zoom Functions allow the appearance of a map feature to change with map's zoom level. Zoom functions can be used to create the illusion of depth and control data density. Each stop is an array with two elements: the first is a zoom level and the second is a function output value.

```
{
  "circle-radius": {
    "stops": [
      [5, 1],
      [10, 2]
    ]
  }
}
```

The rendered values of *color*, *number*, and *array* properties are intepolated between stops. *Enum*, *boolean*, and *string* property values cannot be intepolated, so their rendered values only change at the specified stops.

There is an important difference between the way that zoom functions render for layout and paint properties. Paint properties are continuously re-evaluated whenever the zoom level changes, even fractionally. The rendered value of a paint property will change, for example, as the map moves between zoom levels 4.1 and 4.6. Layout properties, on the other hand, are evaluated only once for each integer zoom level. To continue the prior example: the rendering of a layout property will not change between zoom levels 4.1 and 4.6, no matter what stops are specified; but at zoom level 5, the function will be re-evaluated according to the function, and the property's rendered value will change. (You can include fractional zoom levels in a layout property zoom function, and it will affect the generated values; but, still, the rendering will only change at integer zoom levels.)

Property functions allow the appearance of a map feature to change with its properties. Property functions can be used to visually differentate types of features within the same layer or create data visualizations. Each stop is an array with two elements, the first is a property input value and the second is a function output value. Note that support for property functions is not available across all properties and platforms at this time.

```
{
  "circle-color": {
    "property": "temperature",
    "stops": [
      [0, "blue"],
      [100, "red"]
    ]
  }
}
```

Zoom-and-property functions allow the appearance of a map feature to change with both its properties and zoom. Each stop is an array with two elements, the first is an object with a property input value and a zoom, and the second is a function output value. Note that support for property functions is not yet complete.

```
{
  "circle-radius": {
    "property": "rating",
```

```
                                    "stops": [
                                      [{"zoom": 0, "value": 0}, 0],
                                      [{"zoom": 0, "value": 5}, 5],
                                      [{"zoom": 20, "value": 0}, 0],
                                      [{"zoom": 20, "value": 5}, 20]
                                    ]
                                  }
                                }
```

Filter

A filter selects specific features from a layer. A filter is an array of one of the following forms:

Existential Filters

["has", *key*] *feature[key]* exists

["!has", *key*] *feature[key]* does not exist

Comparison Filters

["==", *key, value*] equality: *feature[key] = value*

["!=", *key, value*] inequality: *feature[key] value*

[">", *key, value*] greater than: *feature[key] > value*

[">=", *key, value*] greater than or equal: *feature[key] value*

["<", *key, value*] less than: *feature[key] < value*

["<=", *key, value*] less than or equal: *feature[key] value*

Set Membership Filters

["in", *key, v0, . . . , vn*] set inclusion: *feature[key] {v0, . . . , vn}*

["!in", *key, v0, . . . , vn*] set exclusion: *feature[key] {v0, . . . , vn}*

Combining Filters

["all", *f0, . . . , fn*] logical AND: *f0 . . . fn*

["any", *f0, . . . , fn*] logical OR: *f0 . . . fn*

["none", *f0, . . . , fn*] logical NOR: *¬f0 . . . ¬fn*

A *key* must be a string that identifies a feature property, or one of the following special keys:

- "$type": the feature type. This key may be used with the "==", "!=", "in", and "!in" operators. Possible values are "Point", "LineString", and "Polygon".
- "$id": the feature identifier. This key may be used with the "==", "!=", "has", "!has", "in", and "!in" operators.

A *value* (and *v0, . . . , vn* for set operators) must be a string, number, or boolean to compare the property value against.

Set membership filters are a compact and efficient way to test whether a field matches any of multiple values.

The comparison and set membership filters implement strictly-typed comparisons; for example, all of the following evaluate to false: 0 < "1", 2 == "2", "true" in [true, false].

The "all", "any", and "none" filter operators are used to create compound filters. The values *f0, ..., fn* must be filter expressions themselves.

```
["==", "$type", "LineString"]
```

This filter requires that the class property of each feature is equal to either "street_major", "street_minor", or "street_limited".

```
["in", "class
↪", "street_major", "street_minor", "street_limited"]
```

The combining filter "all" takes the three other filters that follow it and requires all of them to be true for a feature to be included: a feature must have a class equal to "street_limited", its admin_level must be greater than or equal to 3, and its type cannot be Polygon. You could change the combining filter to "any" to allow features matching any of those criteria to be included - features that are Polygons, but have a different class value, and so on.

```
[
  "all",
  ["==", "class", "street_limited"],
  [">=", "admin_level", 3],
  ["!in", "$type", "Polygon"]
]
```

MapBox Style Grammar

JSON does not allow for comments within the data therefore comments will be noted through the placement of the comment between open < and close > angle brackets. All properties are optional unless otherwise noted as Requried

Root Properties

```
{
  "version": 8, <Required>
  "name": "Mapbox Streets",
  "sprite": "mapbox://sprites/mapbox/streets-v8",
  "glyphs
↪": "mapbox://fonts/mapbox/{fontstack}/{range}.pbf",
  "sources": {...}, <Required>
  "layers": [...] <Required>
}
```

layers

Layers are drawn in the order they appear in the layer array. Layers have two additional properties that determine how data is rendered: *layout* and *paint*

Background Layer definition

```
{
  "layers" : [
    {
      "id": "backgroundcolor",
      "type": "background",
      "source": "test-source",
      "source-layer": "test-source-layer",
      "layout": {
        "visibility": "visible"
      },
      "paint": {
        "background-opacity": 0.45,
        "background-color": "#00FF00"
      }
    }
  ]
}
```

background-color is disabled by the presence of *background-pattern*

Fill Layer Definition

```
{
  "layers": [
    {
      "id": "testid",
      "type": "fill",
      "source": "geoserver-states",
      "source-layer": "states",
      "layout": {
        "visibility": "visible"
      },
      "paint": {
        "fill-antialias": "true",
        "fill-opacity": 0.84,
        "fill-color": "#FF595E",
        "fill-outline-color":"#1982C4",
        "fill-translate": [20,20],
        "fill-translate-anchor": "map",
        "fill-pattern": <String>
      }
    }
  ]
}
```

Line Layer Definition

```
{
  "layers": [
    {
      "id": "test-id",
      "type": "line",
      "source": "test-source",
      "source-layer": "test-source-layer",
      "layout": {
          "line-cap": "square",
          "line-join": "round",
          "line-mitre-
↪limit": 2, <Optional - Requires line-join=mitre>
```

```
                "line-round-
↪limit": 1.05, <Optional - Requires line-join=round>
                "visibility": "visible"
            },
            "paint": {
              "line-color": "#0099ff",
              "line-opacity": 0.5,
              "line-translate": [3,3],
              "line-translate-anchor": "viewport",
              "line-width": 10,
              "line-gap-width": 8,
              "line-offset": 4,
              "line-blur": 2,
              "line-dasharray": [50, 50],
              "line-pattern": <String>
            }
          }
        ],
      }
```

Symbol Layer Definition

```
{
  "layers": [
    {
      "id": "test-id",
      "type": "symbol",
      "source": "test-source",
      "source-layer": "test-source-layer",
      "layout": {
          "symbol-placement
↪": "", <Enum, [Point, line] Defaults to Point>
          "symbol-spacing": "", <Number in pixels.
↪ Defaults to 250, requires symbol-placement = line>
↪ "symbol-avoid-edges": "", <Boolean defaults to true>
↪"icon-allow-overlap": "", <Boolean defaults to false>
          "icon-
↪ignore-placement": "", <Boolean defaults to false>
          "icon-optional": "", <Boolean defaults␣
↪to false, requires icon-image and text-field>
          "icon-rotation-alignment": "", <Enum, [map,␣
↪viewport, auto] defaults to auto requires icon-image>
          "icon-size": "", <Number, defaults to 1>
          "icon-rotation-
↪alignment": "", <Enum, [none, width, height, both]␣
↪defaults to none requires icon-image and text-field>
          "icon-text-fit-padding
↪": "", <Array, units in pixels, defaults to [0,0,0,0]
                requires icon-image, text-field␣
↪and icon-text-fit of one of [both, width, height]>
          "icon-image": "", <String>
          "icon-
↪rotate": "", <Number, in degrees, defaults to 0>
          "icon-padding
↪": "", <Number, units in pixels, defaults to 2>
↪ "icon-keep-upright": "", <Boolean defaults to false,
↪ requires icon-image, icon-rotation-alignment = map
```

```
                    and symbol-placement = line>
            "icon-offset
↪": "", <Array, defaults to [0,0] requires icon-image>
            "text-pitch-alignment": "", <Enum, [map,␣
↪viewport, auto] defaults to auto requires text-field>
            "text-rotation-alignment": "", <Enum, [map,␣
↪viewport, auto] defaults to auto requires text-field>
            "text-field": "", <String>
            "text-
↪font": "", <Array, defaults to [Open Sans Regular,
↪Arial Unicode MS Regular], requires text-field>
            "text-size": "", <Number,␣
↪units in pixels, defaults to 16, requires text-field>
            "text-max-width": "", <Number,
↪ units in ems, defaults to 10 requires text-field>
            "text-line-height": "", <Number,
↪ units in ems, defaults to 1.2 requires text-field>
            "text-letter-spacing": "", <Number,
↪ units in ems, defaults to 0 requires text-field>
            "text-justify": "", <Enum, [left, center,
↪ right] defaults to center requires text-field>
            "text-anchor": "",
↪ <Enum, [center, left, right, top, bottom, top-left,
               top-
↪right, bottom-left, bottom-right] defaults to center>
            "text-max-
↪angle": "", <Number units in degrees, defaults to 45>
            "text-
↪rotate": "", <Number units in degrees, defaults to 0>
            "text-
↪padding": "", <Number units in pixels, defaults to 2>
            ␣
↪ "text-keep-upright": "", <Boolean, defaults to true,
↪ requires text-field, text-rotation-alignment = map,
                and symbol-placement = line>
            "text-transform": "", <Enum [none, uppercase,
↪ lowercase] defaults to none, requires text-field>
            "text-offset": "", <Array,␣
↪units in ems, defaults to [0,0], requires text-field>
            "text-allow-overlap":␣
↪"", <Boolean, defaults to false, requires text-field>
            "text-ignore-placement":␣
↪"", <Boolean, defaults to false, requires text-field>
            "text-optional": "", <Boolean, defaults␣
↪to false, requires text-field and icon-image>
            "visibility": "visible"
        },
        "paint": {
          "icon-opacity": "", <Number, defaults to 1>
          "icon-color":␣
↪"", <Color, defaults to #000000, requires icon-image>
          "icon-halo-color": "", <Color,
↪ defaults to rgba(0,0,0,0) requires icon-image>
          "icon-halo-width": "", <Number,
↪ units in pixels, defaults to 0 requires icon-image>
          "icon-halo-blur": "", <Number,
↪ units in pixels, defaults to 0 requires icon-image>
          "icon-translate": "", <Array, units␣
↪in pixels, defaults to [0,0], requires icon-image>
```

```
                                    "icon-translate-anchor": "", <Enum, [map, viewport],
↪defaults to map, requires icon-image, icon-translate>
                                    "text-opacity
↪": "", <Number, defaults to 1 requires text-field>
                                    "text-halo-color": "", <Color,
↪ defaults to rgba(0,0,0,0) requires text-field>
                                    "text-halo-width": "", <Number,
↪ units in pixels, defaults to 0 requires text-field>
                                    "text-halo-blur": "", <Number,
↪ units in pixels, defaults to 0 requires text-field>
                                    "text-translate": "", <Array units
↪in pixels defaults to [0,0] requires text-field>

↪   "text-translate-anchor": "" <Enum, [map, viewport]
↪defaults to map, requires text-field, text-translate>
                                  }
                                }
                              ],
                            }
```

Raster Layer Definition

```
                            {
                              "layers": [
                                {
                                  "id": "test-id",
                                  "type": "raster",
                                  "source": "test-source",
                                  "source-layer": "test-source-layer",
                                  "layout": {
                                      "visibility": "visible"
                                  },
                                  "paint": {
                                    "raster-opacity": "", <Number defaults to 1>
                                    "raster-hue-
↪rotate": "", <Number units in degrees, defaults to 0>

↪ "raster-brightness-min": "", <Number, defaults to 0>

↪ "raster-brightness-max": "", <Number, defaults to 1>

↪     "raster-saturation": "", <Number, defaults to 0>
                                    "raster-contrast": "", <Number, defaults to 0>
                                    "raster-fade-duration":
↪"" <Number, units in milliseconds, defaults to 300>
                                  }
                                }
                              ],
                            }
```

Circle Layer definition

```
                            {
                              "layers": [
                                {
                                  "id": "test-id",
                                  "type": "raster",
```

```
        "source": "test-source",
        "source-layer": "test-source-layer",
        "layout": {
            "visibility": "visible"
        },
        "paint": {
          "circle-
↪radius": "", <Number, units in pixels, defaults to 5>
↪
↪      "circle-color": "", <Color, defaults to #000000>
          "circle-blur": "", <Number, defaults to 0>
          "circle-opacity": "", <Number, defaults to 1>
          "circle-translate
↪": "", <Array, units in pixels, defaults to [0,0]>
          "circle-translate-anchor": "", <Enum, [map,
↪ viewport] defaults to map requires circle-translate>
          "circle-pitch-
↪scale": "", <Enum, [map, viewport] defaults to map>
          "circle-stroke-
↪width": "", <Number, units in pixels, defaults to 0>
          "circle-
↪stroke-color": "", <Color, defaults to #000000>
↪
↪ "circle-stroke-opacity": "", <Number, defaults to 1>
        }
      }
    ],
}
```

Fill-Extrusion Layer Definition

```
{
  {
    "layers": [
      {
        "id": "test-id",
        "type": "fill-extrusion",
        "source": "test-source",
        "source-layer": "test-source-layer",
        "layout": {
            "visibility": "visible"
        },
        "paint": {
          "
↪"fill-extrusion-opacity": "", <Number, defaults to 1>
          "fill-extrusion-color": "", <Color, defaults
↪to #000000, disabled by fill-extrusion-pattern>
          "fill-extrusion-translate
↪": "", <Array, units in pixels, defaults to [0,0]>
          "fill-extrusion-
↪translate-anchor": "", <Enum, [map, viewport]
↪defaults to map requires fill-extrusion-translate>
          "fill-extrusion-pattern": "", <String>
          "fill-extrusion-
↪height": "", <Number, units in meters, defaults to 0>
↪
↪  "fill-extrusion-base": "" <Number, units in meters,
↪ defaults to 0, requires fill-extrusion-height>
```

```
                }
            }
        ],
    }
```

6.6.4 Mapbox Style Specification

A Mapbox style is a document that defines the visual appearance of a map: what data to draw, the order to draw it in, and how to style the data when drawing it. A style document is a JSON object with specific root level and nested properties. This specification defines and describes these properties.

The intended audience of this quick reference includes:

- Advanced designers and cartographers who want to write styles by hand
- GeoTools developers using the mbstyle module
- Authors of software that generates or processes Mapbox styles.
- Feature support is provided for the Mapbox GL JS, the Open Layers Mapbox Style utility and the GeoTools mbstyle module.
- Where appropriate examples have been changed to reference GeoWebCache.

Note: The Mapbox Style Specification is generated from the BSD Mapbox GL JS github repository, reproduced here with details on this GeoTools implementation.

Root Properties

Root level properties of a Mapbox style specify the map's layers, tile sources and other resources, and default values for the initial camera position when not specified elsewhere.

```
{
    "version": 8,
    "name": "Mapbox Streets",
    "sprite": "sprites/streets-v8",
    "glyphs": "{fontstack}/{range}.pbf",
    "sources": {...},
    "layers": [...]
}
```

version

Required Enum.

Style specification version number. Must be 8.

```
"version": 8
```

name

Optional *String*.

A human-readable name for the style.

```
"name": "Bright"
```

metadata

Optional

Arbitrary properties useful to track with the stylesheet, but do not influence rendering. Properties should be prefixed to avoid collisions.

Note: *unsupported.*

center

Optional *Array*.

Default map center in longitude and latitude. The style center will be used only if the map has not been positioned by other means (e.g. map options or user interaction).

```
"center": [
  -73.9749, 40.7736
]
```

Note: *unsupported*

zoom

Optional *Number*.

Default zoom level. The style zoom will be used only if the map has not been positioned by other means (e.g. map options or user interaction).

```
"zoom": 12.5
```

bearing

Optional Number. *Units in degrees. Defaults to* 0.

Default bearing, in degrees clockwise from true north. The style bearing will be used only if the map has not been positioned by other means (e.g. map options or user interaction).

```
"bearing": 29
```

Note: *unsupported*

pitch

Optional Number. *Units in degrees. Defaults to* 0.

Default pitch, in degrees. Zero is perpendicular to the surface, for a look straight down at the map, while a greater value like 60 looks ahead towards the horizon. The style pitch will be used only if the map has not been positioned by other means (e.g. map options or user interaction).

```
"pitch": 50
```

light

The global light source.

```
"light": {
  "anchor": "viewport",
  "color": "white",
  "intensity": 0.4
}
```

sources

Required Sources.

Data source specifications.

```
"sources": {
  "mapbox-streets": {
    "type": "vector",
    "tiles": [
      "http://localhost:8080/
↪geoserver/gwc/service/wmts?REQUEST=GetTile
          &SERVICE=WMTS&
↪VERSION=1.0.0&LAYER=mapbox:streets&STYLE=

↪&TILEMATRIX=EPSG:900913:{z}&TILEMATRIXSET=EPSG:900913

↪    &FORMAT=application/x-protobuf;type=mapbox-vector
```

```
                &TILECOL={x}&TILEROW={y}"
        ],
        "minZoom": 0,
        "maxZoom": 14
    }
}
```

sprite

Optional *String*.

A base URL for retrieving the sprite image and metadata. The extensions `.png`, `.json` and scale factor `@2x.png` will be automatically appended. This property is required if any layer uses the `background-pattern`, `fill-pattern`, `line-pattern`, `fill-extrusion-pattern`, or `icon-image` properties.

```
"sprite" : "/geoserver/styles/mark"
```

glyphs

Optional *String*.

A URL template for loading signed-distance-field glyph sets in PBF format. The URL must include `{fontstack}` and `{range}` tokens. This property is required if any layer uses the `text-field` layout property.

```
"glyphs": "{fontstack}/{range}.pbf"
```

transition

Required *Transition*.

A global transition definition to use as a default across properties.

```
"transition": {
  "duration": 300,
  "delay": 0
}
```

layers

Required *Array*.

Layers will be drawn in the order of this array.

```
"layers": [
  {
    "id": "water",
    "source": "sf:roads",
```

```
      "source-layer": "water",
      "type": "fill",
      "paint": {
        "fill-color": "#00ffff"
      }
    }
]
```

Light

A style's `light` property provides global light source for that style.

```
"light": {
  "anchor": "viewport",
  "color": "white",
  "intensity": 0.4
}
```

anchor

Optional *Enum*. *One of* map, viewport. *Defaults to* viewport.

Whether extruded geometries are lit relative to the map or viewport.

map The position of the light source is aligned to the rotation of the map.

viewport The position of the light source is aligned to the rotation of the viewport.

```
"anchor": "map"
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.27.0	Not yet supported	Not yet supported

position

Optional *Array*. *Defaults to* 1.15,210,30.

Position of the light source relative to lit (extruded) geometries, in [r radial coordinate, a azimuthal angle, p polar angle] where r indicates the distance from the center of the base of an object to its light, a indicates the position of the light relative to 0° (0° when `light.anchor` is set to `viewport` corresponds to the top of the viewport, or 0° when `light.anchor` is set to `map` corresponds to due north, and degrees proceed clockwise), and p indicates the height of the light (from 0°, directly above, to 180°, directly below).

```
"position": [
  1.5,
  90,
  80
]
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.27.0	Not yet supported	Not yet supported

color

Optional Color. *Defaults to* #ffffff.

Color tint for lighting extruded geometries.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.27.0	Not yet supported	Not yet supported

intensity

Optional Number. *Defaults to* 0.5.

Intensity of lighting (on a scale from 0 to 1). Higher numbers will present as more extreme contrast.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.27.0	Not yet supported	Not yet supported

Sources

Sources supply data to be shown on the map. The type of source is specified by the `"type"` property, and must be one of vector, raster, geojson, image, video, canvas. Adding a source won't immediately make data appear on the map because sources don't contain styling details like color or width. Layers refer to a source and give it a visual representation. This makes it possible to style the same source in different ways, like differentiating between types of roads in a highways layer.

Tiled sources (vector and raster) must specify their details in terms of the TileJSON specification. This can be done in several ways:

- By supplying TileJSON properties such as `"tiles"`, `"minzoom"`, and `"maxzoom"` directly in the source:

```
"mapbox-streets": {
  "type": "vector",
  "tiles": [
    "http://a.example.com/tiles/{z}/{x}/{y}.pbf",
    "http://b.example.com/tiles/{z}/{x}/{y}.pbf"
  ],
  "maxzoom": 14
}
```

- By providing a `"url"` to a TileJSON resource:

```
"mapbox-streets": {
  "type": "vector",
  "url": "http://api.example.com/tilejson.json"
}
```

- By providing a url to a WMS server that supports EPSG:3857 (or EPSG:900913) as a source of tiled data. The server url should contain a `"{bbox-epsg-3857}"` replacement token to supply the `bbox` parameter.

```
"wms-imagery": {
  "type": "raster",
  "tiles": [
  'http://
↪a.example.com/wms?bbox={bbox-epsg-3857}&format=image/
↪png&service=WMS&version=1.1.1&request=GetMap&
↪srs=EPSG:3857&width=256&height=256&layers=example'
  ],
  "tileSize": 256
}
```

vector

A vector tile source. Tiles must be in Mapbox Vector Tile format. All geometric coordinates in vector tiles must be between `-1 * extent` and `(extent * 2) - 1` inclusive. All layers that use a vector source must specify a `"source-layer"` value. For vector tiles hosted by Mapbox, the `"url"` value should be of the form `mapbox://mapid`.

```
"mapbox-streets": {
  "type": "vector",
  "tiles": [
    "http://localhost:8080/geoserver/
↪gwc/service/wmts?REQUEST=GetTile&SERVICE=WMTS
        &VERSION=1.0.0&LAYER=mapbox:streets&
↪STYLE=&TILEMATRIX=EPSG:900913:{z}
        &TILEMATRIXSET=EPSG:900913&
↪FORMAT=application/x-protobuf;type=mapbox-vector
        &TILECOL={x}&TILEROW={y}"
  ],
  "minZoom": 0,
  "maxZoom": 14
}
```

url

Optional String.

A URL to a TileJSON resource. Supported protocols are `http:`, `https:`, and `mapbox://<mapid>`.

tiles

Optional *Array*.

An array of one or more tile source URLs, as in the TileJSON spec.

minzoom

Optional *Number*. *Defaults to* 0.

Minimum zoom level for which tiles are available, as in the TileJSON spec.

maxzoom

Optional *Number*. *Defaults to* 22.

Maximum zoom level for which tiles are available, as in the TileJSON spec. Data from tiles at the maxzoom are used when displaying the map at higher zoom levels.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	Not yet supported	>= 2.4.0

raster

A raster tile source. For raster tiles hosted by Mapbox, the `"url"` value should be of the form `mapbox://mapid`.

```
"mapbox-satellite": {
  "type": "raster",
  "tiles": [
    "http://localhost:8080/geoserver/
↪gwc/service/wmts?REQUEST=GetTile&SERVICE=WMTS
        &VERSION=1.0.0&LAYER=mapbox:satellite&
↪STYLE=&TILEMATRIX=EPSG:900913:{z}
        &TILEMATRIXSET=EPSG:900913&
↪FORMAT=image/png&TILECOL={x}&TILEROW={y}"
  ],
  "minzoom": 0,
  "maxzoom": 14
}
```

url

Optional *String*.

A URL to a TileJSON resource. Supported protocols are `http:`, `https:`, and `mapbox://<mapid>`.

tiles

Optional *Array*.

An array of one or more tile source URLs, as in the TileJSON spec.

minzoom

Optional *Number*. *Defaults to* 0.

Minimum zoom level for which tiles are available, as in the TileJSON spec.

maxzoom

Optional *Number*. *Defaults to* 22.

Maximum zoom level for which tiles are available, as in the TileJSON spec. Data from tiles at the maxzoom are used when displaying the map at higher zoom levels.

tileSize

Optional *Number*. *Defaults to* 512.

The minimum visual size to display tiles for this layer. Only configurable for raster layers.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	Not yet supported	>= 2.4.0

geojson

A GeoJSON source. Data must be provided via a `"data"` property, whose value can be a URL or inline GeoJSON.

```
"geojson-marker": {
  "type": "geojson",
  "data": {
    "type": "Feature",
    "geometry": {
      "type": "Point",
      "coordinates": [-77.0323, 38.9131]
    },
    "properties": {
      "title": "Mapbox DC",
      "marker-symbol": "monument"
    }
  }
}
```

This example of a GeoJSON source refers to an external GeoJSON document via its URL. The GeoJSON document must be on the same domain or accessible using CORS.

```
"geojson-lines": {
  "type": "geojson",
  "data": "./lines.geojson"
}
```

data

Optional

A URL to a GeoJSON file, or inline GeoJSON.

maxzoom

Optional *Number*. *Defaults to* 18.

Maximum zoom level at which to create vector tiles (higher means greater detail at high zoom levels).

buffer

Optional *Number*. *Defaults to* 128.

Size of the tile buffer on each side. A value of 0 produces no buffer. A value of 512 produces a buffer as wide as the tile itself. Larger values produce fewer rendering artifacts near tile edges and slower performance.

tolerance

Optional *Number*. *Defaults to* 0.375.

Douglas-Peucker simplification tolerance (higher means simpler geometries and faster performance).

cluster

Optional *Boolean*. *Defaults to* false.

If the data is a collection of point features, setting this to true clusters the points by radius into groups.

clusterRadius

Optional *Number*. *Defaults to* 50.

Radius of each cluster if clustering is enabled. A value of 512 indicates a radius equal to the width of a tile.

clusterMaxZoom

Optional *Number*.

Max zoom on which to cluster points if clustering is enabled. Defaults to one zoom less than maxzoom (so that last zoom features are not clustered).

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	Not yet supported	>= 2.4.0
clustering	>= 0.14.0	Not yet supported	Not yet supported

image

An image source. The `"url"` value contains the image location.

The `"coordinates"` array contains `[longitude, latitude]` pairs for the image corners listed in clockwise order: top left, top right, bottom right, bottom left.

```
"image": {
  "type": "image",
  "url": "/mapbox-gl-js/assets/radar.gif",
  "coordinates": [
      [-80.425, 46.437],
      [-71.516, 46.437],
      [-71.516, 37.936],
      [-80.425, 37.936]
  ]
}
```

url

Required *String*.

URL that points to an image.

coordinates

Required *Array*.

Corners of image specified in longitude, latitude pairs.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	Not yet supported	Not yet supported

video

A video source. The `"urls"` value is an array. For each URL in the array, a video element source will be created, in order to support same media in multiple formats supported by different browsers.

The `"coordinates"` array contains `[longitude, latitude]` pairs for the video corners listed in clockwise order: top left, top right, bottom right, bottom left.

```
"video": {
  "type": "video",
  "urls": [
    "https://www.mapbox.com/drone/video/drone.mp4",
    "https://www.mapbox.com/drone/video/drone.webm"
  ],
  "coordinates": [
     [-122.51596391201019, 37.56238816766053],
     [-122.51467645168304, 37.56410183312965],
     [-122.51309394836426, 37.563391708549425],
     [-122.51423120498657, 37.56161849366671]
  ]
}
```

urls

Required *Array*.

URLs to video content in order of preferred format.

coordinates

Required *Array*.

Corners of video specified in longitude, latitude pairs.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	Not yet supported	Not yet supported

canvas

A canvas source. The `"canvas"` value is the ID of the canvas element in the document.

The `"coordinates"` array contains `[longitude, latitude]` pairs for the video corners listed in clockwise order: top left, top right, bottom right, bottom left.

If an HTML document contains a canvas such as this:

```
<canvas id="mycanvas" width=
↪"400" height="300" style="display: none;"></canvas>
```

the corresponding canvas source would be specified as follows:

```
"canvas": {
  "type": "canvas",
  "canvas": "mycanvas",
  "coordinates": [
     [-122.51596391201019, 37.56238816766053],
     [-122.51467645168304, 37.56410183312965],
     [-122.51309394836426, 37.563391708549425],
     [-122.51423120498657, 37.56161849366671]
  ]
}
```

coordinates

Required *Array*.

Corners of canvas specified in longitude, latitude pairs.

animate

Whether the canvas source is animated. If the canvas is static, `animate` should be set to `false` to improve performance.

canvas

Required *String*.

HTML ID of the canvas from which to read pixels.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.32.0	Not yet supported	Not yet yupported

Sprite

A style's `sprite` property supplies a URL template for loading small images to use in rendering `background-pattern`, `fill-pattern`, `line-pattern`, and `icon-image` style properties.

```
"sprite" : "/geoserver/styles/mark"
```

A valid sprite source must supply two types of files:

- An *index file*, which is a JSON document containing a description of each image contained in the sprite. The content of this file must be a JSON object whose keys form identifiers to be used as the values of the above style properties, and whose values are objects describing the dimensions (`width` and `height` properties) and pixel ratio (`pixelRatio`) of the image and its location within the sprite (`x` and

y). For example, a sprite containing a single image might have the following index file contents:

```
{
  "poi": {
    "width": 32,
    "height": 32,
    "x": 0,
    "y": 0,
    "pixelRatio": 1
  }
}
```

Then the style could refer to this sprite image by creating a symbol layer with the layout property `"icon-image": "poi"`, or with the tokenized value `"icon-image": "{icon}"` and vector tile features with a `icon` property with the value `poi`.

- *Image files*, which are PNG images containing the sprite data.

  Mapbox SDKs will use the value of the `sprite` property in the style to generate the URLs for loading both files. First, for both file types, it will append `@2x` to the URL on high-DPI devices. Second, it will append a file extension: `.json` for the index file, and `.png` for the image file. For example, if you specified `"sprite": "https://example.com/sprite"`, renderers would load `https://example.com/sprite.json` and `https://example.com/sprite.png`, or `https://example.com/sprite@2x.json` and `https://example.com/sprite@2x.png`.

  If you are using Mapbox Studio, you will use prebuilt sprites provided by Mapbox, or you can upload custom SVG images to build your own sprite. In either case, the sprite will be built automatically and supplied by Mapbox APIs. If you want to build a sprite by hand and self-host the files, you can use spritezero-cli, a command line utility that builds Mapbox GL compatible sprite PNGs and index files from a directory of SVGs.

Glyphs

A style's `glyphs` property provides a URL template for loading signed-distance-field glyph sets in PBF format.

```
"glyphs": "{fontstack}/{range}.pbf"
```

This URL template should include two tokens:

- `{fontstack}` When requesting glyphs, this token is replaced with a comma separated list of fonts from a font stack specified in the `text-font <#layout-symbol-text-font>'__ property of a symbol layer.
- `{range}` When requesting glyphs, this token is replaced with a range of 256 Unicode code points. For example, to load glyphs for the Unicode Basic Latin and Basic Latin-1 Supplement blocks, the range would be `0-255`. The actual ranges that are loaded are determined at runtime based on what text needs to be displayed.

Transition

A style's `transition` property provides global transition defaults for that style.

```
"transition": {
  "duration": 300,
  "delay": 0
}
```

duration

Optional Number. Units in milliseconds. Defaults to 300.

Time allotted for transitions to complete.

delay

Optional Number. Units in milliseconds. Defaults to 0.

Length of time before a transition begins.

Layers

A style's `layers` property lists all of the layers available in that style. The type of layer is specified by the `"type"` property, and must be one of background, fill, line, symbol, raster, circle, fill-extrusion.

Except for layers of the background type, each layer needs to refer to a source. Layers take the data that they get from a source, optionally filter features, and then define how those features are styled.

```
"layers": [
  {
    "id": "water",
    "source": "sf:roads",
    "source-layer": "water",
    "type": "fill",
    "paint": {
      "fill-color": "#00ffff"
    }
  }
]
```

Layer Properties

id

Required String.

Unique layer name.

type

Optional Enum. One of fill, line, symbol, circle, fill-extrusion, raster, background.

Rendering type of this layer.

fill A filled polygon with an optional stroked border.

line A stroked line.

symbol An icon or a text label.

circle A filled circle.

fill-extrusion An extruded (3D) polygon.

raster Raster map textures such as satellite imagery.

background The background color or pattern of the map.

metadata

Optional

Arbitrary properties useful to track with the layer, but do not influence rendering. Properties should be prefixed to avoid collisions, like 'mapbox:'.

source

Optional String.

Name of a source description to be used for this layer.

source-layer

Optional String.

Layer to use from a vector tile source. Required if the source supports multiple layers.

minzoom

Optional Number.

The minimum zoom level on which the layer gets parsed and appears on.

maxzoom

Optional Number.

The maximum zoom level on which the layer gets parsed and appears on.

filter

Optional *Expression*.

A expression specifying conditions on source features. Only features that match the filter are displayed.

layout

layout properties for the layer

paint

Optional paint properties for the layer

Layers have two sub-properties that determine how data from that layer is rendered: `layout` and `paint` properties.

Layout properties appear in the layer's `"layout"` object. They are applied early in the rendering process and define how data for that layer is passed to the GPU. For efficiency, a layer can share layout properties with another layer via the `"ref"` layer property, and should do so where possible. This will decrease processing time and allow the two layers will share GPU memory and other resources associated with the layer.

Paint properties are applied later in the rendering process. A layer that shares layout properties with another layer can have independent paint properties. Paint properties appear in the layer's `"paint"` object.

background

Layout Properties

visibility

Optional *Enum*. *One of* visible, none, *Defaults to* visible.

Whether this layer is displayed.

visible The layer is shown.

none The layer is not shown.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	Not yet supported	>= 2.4.0

Paint Properties

background-color

Optional Color. *Defaults to* #000000. *Disabled by* background-pattern.

The color with which the background will be drawn.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	Not yet supported	>= 2.4.0

background-pattern

Optional String.

Name of image in sprite to use for drawing an image background. For seamless patterns, image width and height must be a factor of two (2, 4, 8, . . . , 512).

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	Not yet supported	Not yet supported

background-opacity

Optional Number. *Defaults to* 1.

The opacity at which the background will be drawn.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	Not yet supported	>= 2.4.0

fill

Layout Properties

visibility

Optional Enum. *One of* visible, none. *Defaults to* visible.

Whether this layer is displayed.

visible The layer is shown.

none The layer is not shown.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	>= 17.1	>= 2.4.0

Paint Properties

fill-antialias

Optional Boolean. *Defaults to* true.

Whether or not the fill should be antialiased.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	Not yet supported	Not yet supported
data-driven styling	Not yet supported	Not yet supported	Not yet supported

fill-opacity

Optional Number. *Defaults to* 1.

The opacity of the entire fill layer. In contrast to the `fill-color`, this value will also affect the 1px stroke around the fill, if the stroke is used.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	>= 17.1	>= 2.4.0
data-driven styling	>= 0.21.0	>= 17.1	>= 2.4.0

fill-color

Optional Color. *Defaults to* #000000. *Disabled by* fill-pattern.

The color of the filled part of this layer. This color can be specified as `rgba` with an alpha component and the color's opacity will not affect the opacity of the 1px stroke, if it is used.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	>= 17.1	>= 2.4.0
data-driven styling	>= 0.19.0	>= 17.1	>= 2.4.0

fill-outline-color

Optional Color. *Disabled by* fill-pattern. *Requires* fill-antialias = true.

The outline color of the fill. Matches the value of `fill-color` if unspecified.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	>= 17.1	>= 2.4.0
data-driven styling	>= 0.19.0	>= 17.1	>= 2.4.0

fill-translate

Optional *Array*. *Units in* pixels. *Defaults to* 0.0.

The geometry's offset. Values are [x, y] where negatives indicate left and up, respectively.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	Not yet supported	Not yet supported
data-driven styling	Not yet supported	Not yet supported	Not yet supported

fill-translate-anchor

Optional *Enum*. *One of* map, viewport. *Defaults to* map. *Requires* fill-translate.

Controls the translation reference point.

map The fill is translated relative to the map.

viewport The fill is translated relative to the viewport.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	Not yet supported	Not yet supported
data-driven styling	Not yet supported	Not yet supported	Not yet supported

fill-pattern

Optional *String*.

Name of image in sprite to use for drawing image fills. For seamless patterns, image width and height must be a factor of two (2, 4, 8, . . . , 512).

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	>= 17.1	Not yet supported
data-driven styling	Not yet supported	>= 17.1	Not yet supported

line

Layout Properties

line-cap

Optional Enum. *One of* butt, round, square. *Defaults to* butt.

The display of line endings.

butt A cap with a squared-off end which is drawn to the exact endpoint of the line.

round A cap with a rounded end which is drawn beyond the endpoint of the line at a radius of one-half of the line's width and centered on the endpoint of the line.

square A cap with a squared-off end which is drawn beyond the endpoint of the line at a distance of one-half of the line's width.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	>= 17.1	>= 2.4.0
data-driven styling	Not yet supported	>= 17.1	>= 2.4.0

line-join

Optional Enum. *One of* bevel, round, miter. *Defaults to* miter.

The display of lines when joining.

bevel A join with a squared-off end which is drawn beyond the endpoint of the line at a distance of one-half of the line's width.

round A join with a rounded end which is drawn beyond the endpoint of the line at a radius of one-half of the line's width and centered on the endpoint of the line.

miter A join with a sharp, angled corner which is drawn with the outer sides beyond the endpoint of the path until they meet.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	>= 17.1	>= 2.4.0
data-driven styling	Not yet supported	>= 17.1	>= 2.4.0

line-miter-limit

Optional Number. *Defaults to* 2. *Requires* line-join = miter.

Used to automatically convert miter joins to bevel joins for sharp angles.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	Not yet supported	>= 2.4.0
data-driven styling	Not yet supported	Not yet supported	>= 2.4.0

line-round-limit

Optional Number. *Defaults to* 1.05. *Requires* line-join = round.

Used to automatically convert round joins to miter joins for shallow angles.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	Not yet supported	Not yet supported
data-driven styling	Not yet supported	Not yet supported	Not yet supported

visibility

Optional Enum. *One of* visible, none. *Defaults to* visible.

Whether this layer is displayed.

visible The layer is shown.

none The layer is not shown.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	>= 17.1	>= 2.4.0
data-driven styling	Not yet supported	>= 17.1	>= 2.4.0

Paint Properties

line-opacity

Optional Number. *Defaults to* 1.

The opacity at which the line will be drawn.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	>= 17.1	>= 2.4.0
data-driven styling	>= 0.29.0	>= 17.1	>= 2.4.0

line-color

Optional Color. *Defaults to* #000000. *Disabled by* line-pattern.

The color with which the line will be drawn.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	>= 17.1	>= 2.4.0
data-driven styling	>= 0.23.0	>= 17.1	>= 2.4.0

line-translate

Optional Array. *Units in* pixels. *Defaults to* 0.0.

The geometry's offset. Values are [x, y] where negatives indicate left and up, respectively.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	Not yet supported	Not yet supported
data-driven styling	Not yet supported	Not yet supported	Not yet supported

line-translate-anchor

Optional Enum. *One of* map, viewport. *Defaults to* map. *Requires* line-translate.

Controls the translation reference point.

map The line is translated relative to the map.

viewport The line is translated relative to the viewport.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	Not yet supported	Not yet supported
data-driven styling	Not yet supported	Not yet supported	Not yet supported

line-width

Optional Number. *Units in* pixels. *Defaults to* 1.

Stroke thickness.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	>= 17.1	>= 2.4.0
data-driven styling	Not yet supported	>= 17.1	>= 2.4.0

line-gap-width

Optional Number. *Units in* pixels. *Defaults to* 0.

Draws a line casing outside of a line's actual path. Value indicates the width of the inner gap.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	Not yet supported	Not yet supported
data-driven styling	>= 0.29.0	Not yet supported	Not yet supported

line-offset

Optional Number. *Units in* pixels. *Defaults to* 0.

The line's offset. For linear features, a positive value offsets the line to the right, relative to the direction of the line, and a negative value to the left. For polygon features, a positive value results in an inset, and a negative value results in an outset.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.12.1	>= 17.1	Not yet supported
data-driven styling	>= 0.29.0	>= 17.1	Not yet supported

line-blur

Optional Number. *Units in* pixels. *Defaults to* 0.

Blur applied to the line, in pixels.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	Not yet supported	Not yet supported
data-driven styling	>= 0.29.0	Not yet supported	Not yet supported

line-dasharray

Optional Array. *Units in* line widths. *Disabled by* line-pattern.

Specifies the lengths of the alternating dashes and gaps that form the dash pattern. The lengths are later scaled by the line width. To convert a dash length to pixels, multiply the length by the current line width.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	>= 17.1	>= 2.4.0
data-driven styling	Not yet supported	>= 17.1	>= 2.4.0

line-pattern

Optional *String*.

Name of image in sprite to use for drawing image lines. For seamless patterns, image width must be a factor of two (2, 4, 8, ..., 512).

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	>= 17.1	Not yet supported
data-driven styling	Not yet supported	>= 17.1	Not yet supported

symbol

Layout Properties

symbol-placement

Optional *Enum*. *One of* point, line. *Defaults to* point.

Label placement relative to its geometry.

point The label is placed at the point where the geometry is located.

line The label is placed along the line of the geometry. Can only be used on `LineString` and `Polygon` geometries.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	>= 17.1	>= 2.10.0
data-driven styling	Not yet supported	>= 17.1	>= 2.10.0

symbol-spacing

Optional *Number*. *Units in* pixels. *Defaults to* 250. *Requires* symbol-placement = line.

Distance between two symbol anchors.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	>= 17.1	Not yet supported
data-driven styling	Not yet supported	Not yet supported	Not yet supported

symbol-avoid-edges

Optional Boolean. *Defaults to* false.

If true, the symbols will not cross tile edges to avoid mutual collisions. Recommended in layers that don't have enough padding in the vector tile to prevent collisions, or if it is a point symbol layer placed after a line symbol layer.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	Not yet supported	Not yet supported
data-driven styling	Not yet supported	Not yet supported	Not yet supported

icon-allow-overlap

Optional Boolean. *Defaults to* false. *Requires* icon-image.

If true, the icon will be visible even if it collides with other previously drawn symbols.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	Not yet supported	Not yet supported
data-driven styling	Not yet supported	Not yet supported	Not yet supported

icon-ignore-placement

Optional Boolean. *Defaults to* false. *Requires* icon-image.

If true, other symbols can be visible even if they collide with the icon.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	Not yet supported	Not yet supported
data-driven styling	Not yet supported	Not yet supported	Not yet supported

icon-optional

Optional Boolean. *Defaults to* false. <*Requires* icon-image, text-field.

If true, text will display without their corresponding icons when the icon collides with other symbols and the text does not.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	Not yet supported	Not yet supported
data-driven styling	Not yet supported	Not yet supported	Not yet supported

icon-rotation-alignment

Optional Enum. *One of* map, viewport, auto. *Defaults to* auto. *Requires* icon-image.

In combination with `symbol-placement`, determines the rotation behavior of icons.

map When `symbol-placement` is set to `point`, aligns icons east-west. When `symbol-placement` is set to `line`, aligns icon x-axes with the line.

viewport Produces icons whose x-axes are aligned with the x-axis of the viewport, regardless of the value of `symbol-placement`.

auto When `symbol-placement` is set to `point`, this is equivalent to `viewport`. When `symbol-placement` is set to `line`, this is equivalent to `map`.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	Not yet supported	Not yet supported
`auto` value	>= 0.25.0	Not yet supported	Not yet supported
data-driven styling	Not yet supported	Not yet supported	Not yet supported

icon-size

Optional Number. *Defaults to* 1. *Requires* icon-image. Scale factor for icon. 1 is original size, 3 triples the size.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	Not yet supported	>= 2.4.0
data-driven styling	>= 0.35.0	Not yet supported	>= 2.4.0

icon-text-fit

Optional Enum. *One of* none, width, height, both. *Defaults to* none. *Requires* icon-image, text-field.

Scales the icon to fit around the associated text.

none The icon is displayed at its intrinsic aspect ratio.

width The icon is scaled in the x-dimension to fit the width of the text.

height The icon is scaled in the y-dimension to fit the height of the text.

both The icon is scaled in both x- and y-dimensions.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.21.0	>= 17.1	Not yet supported
data-driven styling	Not yet supported	Not yet supported	Not yet supported

icon-text-fit-padding

*Optional :ref:'types-array'. *Units in* pixels. *Defaults to* 0,0,0,0. *Requires* icon-image, text-field, icon-text-fit = one of both, width, height.

Size of the additional area added to dimensions determined by `icon-text-fit`, in clockwise order: top, right, bottom, left.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.21.0	>= 17.1	Not yet supported
data-driven styling	Not yet supported	Not yet supported	Not yet supported

icon-image

Optional String.

Name of image in sprite to use for drawing an image background. A string with {tokens} replaced, referencing the data property to pull from.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	>= 17.1	>= 2.4.0
data-driven styling	Not yet supported	>= 17.1	>= 2.4.0

icon-rotate

Optional Number. *Units in* degrees. *Defaults to* 0. *Requires* icon-image.

Rotates the icon clockwise.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	>= 17.1	>= 2.4.0
data-driven styling	>= 0.21.0	>= 17.1	>= 2.4.0

icon-padding

Optional Number. *Units in* pixels. *Defaults to* 2. *Requires* icon-image.

Size of the additional area around the icon bounding box used for detecting symbol collisions.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	Not yet supported	Not yet supported
data-driven styling	Not yet supported	Not yet supported	Not yet supported

icon-keep-upright

Optional Boolean. *Defaults to* false. *Requires* icon-image, icon-rotation-alignment = map, symbol-placement = line.

If true, the icon may be flipped to prevent it from being rendered upside-down.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	Not yet supported	Not yet supported
data-driven styling	Not yet supported	Not yet supported	Not yet supported

icon-offset

Optional Array. *Defaults to* 0,0. *Requires* icon-image.

Offset distance of icon from its anchor. Positive values indicate right and down, while negative values indicate left and up. When combined with `icon-rotate` the offset will be as if the rotated direction was up.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	>= Not yet supported	Not yet supported
data-driven styling	>= 0.29.0	>= Not yet supported	Not yet supported

text-pitch-alignment

Optional Enum *One of* map, viewport, auto. *Defaults to* auto. *Requires* text-field.

Orientation of text when map is pitched.

map The text is aligned to the plane of the map.

viewport The text is aligned to the plane of the viewport.

auto Automatically matches the value of `text-rotation-alignment`.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	Not yet supported	Not yet supported
`auto` value	>= 0.25.0	Not yet supported	Not yet supported
data-driven styling	Not yet supported	Not yet supported	Not yet supported

text-rotation-alignment

Optional Enum. *One of* map, viewport, auto. *Defaults to* auto. *Requires* text-field.

In combination with `symbol-placement`, determines the rotation behavior of the individual glyphs forming the text.

map When `symbol-placement` is set to `point`, aligns text east-west. When `symbol-placement` is set to `line`, aligns text x-axes with the line.

viewport Produces glyphs whose x-axes are aligned with the x-axis of the viewport, regardless of the value of `symbol-placement`.

auto When `symbol-placement` is set to `point`, this is equivalent to `viewport`. When `symbol-placement` is set to `line`, this is equivalent to `map`.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	Not yet supported	Not yet supported
`auto` value	>= 0.25.0	Not yet supported	
data-driven styling	Not yet supported	Not yet supported	Not yet supported

text-field

Optional String.

Value to use for a text label. Feature properties are specified using tokens like {field_name}. (Token replacement is only supported for literal `text-field` values–not for property functions.)

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	>= 17.1	>= 2.4.0
data-driven styling	>= 0.33.0	>= 17.1	>= 2.4.0

text-font

Optional *Array*. *Defaults to* Open Sans Regular,Arial Unicode MS Regular. *Requires* text-field.

Font stack to use for displaying text.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	>= 17.1	>= 2.4.0
data-driven styling	Not yet supported	Not yet supported	>= 2.4.0

text-size

Optional *Number*. *Units in* pixels. *Defaults to* 16. *Requires* text-field.

Font size.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	>= 17.1	>= 2.4.0
data-driven styling	>= 0.35.0	>= 17.1	>= 2.4.0

text-max-width

Optional *Number*. *Units in* pixels. *Defaults to* 10. *Requires* text-field.

The maximum line width for text wrapping.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	Not yet supported	>= 2.4.0
data-driven styling	Not yet supported	Not yet supported	>= 2.4.0

text-line-height

Optional *Number*. *Units in* ems. *Defaults to* 1.2. *Requires* text-field.

Text leading value for multi-line text.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	Not yet supported	Not yet supported
data-driven styling	Not yet supported	Not yet supported	Not yet supported

text-letter-spacing

Optional Number. *Units in* ems. *Defaults to* 0. *Requires* text-field.

Text tracking amount.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	Not yet supported	Not yet supported
data-driven styling	Not yet supported	Not yet supported	Not yet supported

text-justify

Optional Enum. *One of* left, center, right. *Defaults to* center. *Requires* text-field.

Text justification options.

left The text is aligned to the left.

center The text is centered.

right The text is aligned to the right.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	Not yet supported	Not yet supported
data-driven styling	Not yet supported	Not yet supported	Not yet supported

text-anchor

Optional Enum. *One of* center, left, right, top, bottom, top-left, top-right, bottom-left, bottom-right. *Defaults to* center. *Requires* text-field.

Part of the text placed closest to the anchor.

center The center of the text is placed closest to the anchor.

left The left side of the text is placed closest to the anchor.

right The right side of the text is placed closest to the anchor.

top The top of the text is placed closest to the anchor.

bottom The bottom of the text is placed closest to the anchor.

top-left The top left corner of the text is placed closest to the anchor.

top-right The top right corner of the text is placed closest to the anchor.

bottom-left The bottom left corner of the text is placed closest to the anchor.

bottom-right The bottom right corner of the text is placed closest to the anchor.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	Not yet supported	>= 2.4.0
data-driven styling	Not yet supported	Not yet supported	>= 2.4.0

text-max-angle

Optional Number. *Units in* degrees. *Defaults to* 45. *Requires* text-field, symbol-placement = line.

Maximum angle change between adjacent characters.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	Not yet supported	>= 2.10.0
data-driven styling	Not yet supported	Not yet supported	>= 2.10.0

text-rotate

Optional Number. *Units in* degrees. *Defaults to* 0. *Requires* text-field.

Rotates the text clockwise.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	Not yet supported	>= 2.10.0
data-driven styling	>= 0.35.0	Not yet supported	>= 2.10.0

text-padding

Optional Number. *Units in* pixels. *Defaults to* 2. *Requires* text-field.

Size of the additional area around the text bounding box used for detecting symbol collisions.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	Not yet supported	Not yet supported
data-driven styling	Not yet supported	Not yet supported	Not yet supported

text-keep-upright

Optional Boolean. *Defaults to* true. *Requires* text-field, text-rotation-alignment = true, symbol-placement = true.

If true, the text may be flipped vertically to prevent it from being rendered upside-down.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	Not yet supported	Not yet supported
data-driven styling	Not yet supported	Not yet supported	Not yet supported

text-transform

Optional Enum. *One of* none, uppercase, lowercase. *Defaults to* none. *Requires* text-field.

Specifies how to capitalize text, similar to the CSS `text-transform` property.

none The text is not altered.

uppercase Forces all letters to be displayed in uppercase.

lowercase Forces all letters to be displayed in lowercase.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	Not yet supported	>= 2.4.0
data-driven styling	>= 0.33.0	Not yet supported	>= 2.4.0

text-offset

Optional Array. *Units in* pixels. *Defaults to* 0,0. *Requires* icon-image.

Offset distance of text from its anchor. Positive values indicate right and down, while negative values indicate left and up.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	Not yet supported	>= 2.4.0
data-driven styling	>= 0.35.0	Not yet supported	>= 2.4.0

text-allow-overlap

Optional Boolean. *Defaults to* false. *Requires* text-field.

If true, the text will be visible even if it collides with other previously drawn symbols.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	>= 17.1	Not yet supported
data-driven styling	Not yet supported	Not yet supported	Not yet supported

text-ignore-placement

Optional Boolean. *Defaults to* false. *Requires* text-field

If true, other symbols can be visible even if they collide with the text.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	Not yet supported	Not yet supported
data-driven styling	Not yet supported	Not yet supported	Not yet supported

text-optional

Optional Boolean. *Defaults to* false. *Requires* text-field, icon-image.

If true, icons will display without their corresponding text when the text collides with other symbols and the icon does not.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	Not yet supported	Not yet supported
data-driven styling	Not yet supported	Not yet supported	Not yet supported

visibility

Optional Enum. *One of* visible, none. *Defaults to* visible.

Whether this layer is displayed.

visible The layer is shown.

none The layer is not shown.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	>= 17.1	>= 2.4.0
data-driven styling	Not yet supported	>= 17.1	>= 2.4.0

Paint Properties

icon-opacity

Optional Number. *Defaults to* 1. Requires icon-image.

The opacity at which the icon will be drawn.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	>= 17.1	>= 2.4.0
data-driven styling	>= 0.33.0	>= 17.1	>= 2.4.0

icon-color

Optional Color. *Defaults to* #000000. *Requires* icon-image.

The color of the icon. This can only be used with sdf icons.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	Not yet supported	>= 2.10.0
data-driven styling	>= 0.33.0	Not yet supported	>= 2.10.0

icon-halo-color

Optional Color. *Defaults to* rgba(0, 0, 0, 0). *Requires* icon-image.

The color of the icon's halo. Icon halos can only be used with SDF icons.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	Not yet supported	Not yet supported
data-driven styling	>= 0.33.0	Not yet supported	Not yet supported

icon-halo-width

Optional Number. *Units in* pixels. *Defaults to* 0. *Requires* icon-image.

Distance of halo to the icon outline.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	Not yet supported	Not yet supported
data-driven styling	>= 0.33.0	Not yet supported	Not yet supported

icon-halo-blur

Optional Number. *Units in* pixels. *Defaults to* 0. *Requires* icon-image.

Fade out the halo towards the outside.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	Not yet supported	Not yet supported
data-driven styling	>= 0.33.0	Not yet supported	Not yet supported

icon-translate

Optional Array. *Units in* pixels. *Defaults to* 0,0. *Requires* icon-image.

Distance that the icon's anchor is moved from its original placement. Positive values indicate right and down, while negative values indicate left and up.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	Not yet supported	Not yet supported
data-driven styling	Not yet supported	Not yet supported	Not yet supported

icon-translate-anchor

Optional Enum *One of* map, viewport. *Defaults to* map. *Requires* icon-image, icon-translate.

Controls the translation reference point.

map Icons are translated relative to the map.

viewport Icons are translated relative to the viewport.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	Not yet supported	Not yet supported
data-driven styling	Not yet supported	Not yet supported	Not yet supported

text-opacity

Optional Number. *Defaults to* 1. Requires text-field.

The opacity at which the text will be drawn.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	>= 17.1	Not yet supported
data-driven styling	>= 0.33.0	>= 17.1	Not yet supported

text-color

Optional Color. *Defaults to* #000000. *Requires* text-field.

The color with which the text will be drawn.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	>= 17.1	>= 2.4.0
data-driven styling	>= 0.33.0	>= 17.1	>= 2.4.0

text-halo-color

Optional Color. *Defaults to* rgba(0, 0, 0, 0). *Requires* text-field.

The color of the text's halo, which helps it stand out from backgrounds.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	>= 17.1	>= 2.4.0
data-driven styling	>= 0.33.0	>= 17.1	>= 2.4.0

text-halo-width

Optional Number. *Units in* pixels. *Defaults to* 0. *Requires* text-field.

Distance of halo to the font outline. Max text halo width is 1/4 of the font-size.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	>= 17.1	>= 2.4.0
data-driven styling	>= 0.33.0	>= 17.1	>= 2.4.0

text-halo-blur

Optional Number. *Units in* pixels. *Defaults to* 0. *Requires* text-field.

The halo's fadeout distance towards the outside.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	Not yet supported	Not yet supported
data-driven styling	>= 0.33.0	Not yet supported	Not yet supported

text-translate

Optional Array. *Units in* pixels. *Defaults to* 0,0. *Requires* text-field.

Distance that the text's anchor is moved from its original placement. Positive values indicate right and down, while negative values indicate left and up.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	Not yet supported	Not yet supported
data-driven styling	Not yet supported	Not yet supported	Not yet supported

text-translate-anchor

Optional Enum *One of* map, viewport. *Defaults to* map. *Requires* text-field, text-translate.

Controls the translation reference point.

map The text is translated relative to the map.

viewport The text is translated relative to the viewport.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	Not yet supported	Not yet supported
data-driven styling	Not yet supported	Not yet supported	Not yet supported

raster

Layout Properties

visibility

Optional Enum. *One of* visible, none. *Defaults to* visible.

Whether this layer is displayed.

visible The layer is shown.

none The layer is not shown.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	>= 17.1	Not yet supported
data-driven styling	Not yet supported	>= 17.1	Not yet supported

Paint Properties

raster-opacity

Optional Number. *Defaults to* 1.

The opacity at which the image will be drawn.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	>= 17.1	Not yet supported
data-driven styling	Not yet supported	>= 17.1	Not yet supported

raster-hue-rotate

Optional Number. *Units in* degrees. *Defaults to* 0.

Rotates hues around the color wheel.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	Not yet supported	Not yet supported
data-driven styling	Not yet supported	Not yet supported	Not yet supported

raster-brightness-min

Optional Number. *Defaults to* 0.

Increase or reduce the brightness of the image. The value is the minimum brightness.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	Not yet supported	Not yet supported
data-driven styling	Not yet supported	Not yet supported	Not yet supported

raster-brightness-max

Optional Number. *Defaults to* 1.

Increase or reduce the brightness of the image. The value is the maximum brightness.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	Not yet supported	Not yet supported
data-driven styling	Not yet supported	Not yet supported	Not yet supported

raster-saturation

Optional Number. *Defaults to* 0.

Increase or reduce the saturation of the image.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	Not yet supported	Not yet supported
data-driven styling	Not yet supported	Not yet supported	Not yet supported

raster-contrast

Optional Number. *Defaults to* 0.

Increase or reduce the contrast of the image.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	Not yet supported	Not yet supported
data-driven styling	Not yet supported	Not yet supported	Not yet supported

raster-fade-duration

Optional Number *Units in* milliseconds. *Defaults to* 300.

Fade duration when a new tile is added.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	Not yet supported	Not yet supported
data-driven styling	Not yet supported	Not yet supported	Not yet supported

circle

Layout Properties

visibility

Optional Enum. *One of* visible, none. *Defaults to* visible.

Whether this layer is displayed.

visible The layer is shown.

none The layer is not shown.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	>= 17.1	>= 2.4.0

Paint Properties

circle-radius

Optional Number. *Units in* pixels. *Defaults to* 5.

Circle radius.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	>= 17.1	>= 2.4.0
data-driven styling	>= 0.18.0	>= 17.1	>= 2.4.0

circle-color

Optional Color. *Defaults to* #000000.

The fill color of the circle.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	>= 17.1	>= 2.4.0
data-driven styling	>= 0.18.0	>= 17.1	>= 2.4.0

circle-blur

Optional Number. *Defaults to* 0.

Amount to blur the circle. 1 blurs the circle such that only the centerpoint is full opacity.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	Not yet supported	Not yet supported
data-driven styling	>= 0.20.0	Not yet supported	Not yet supported

circle-opacity

Optional Number. *Defaults to* 1.

The opacity at which the circle will be drawn.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	>= 17.1	>= 2.10.0
data-driven styling	>= 0.20.0	>= 17.1	>= 2.10.0

circle-translate

Optional Array. *Units in* pixels. *Defaults to* 0,0.

The geometry's offset. Values are [x, y] where negatives indicate left and up, respectively.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	>= 17.1	Not yet supported
data-driven styling	Not yet supported	>= 17.1	Not yet supported

circle-translate-anchor

Optional Enum *One of* map, viewport. *Defaults to* map. *Requires* circle-translate.

Controls the translation reference point.

map The circle is translated relative to the map.

viewport The circle is translated relative to the viewport.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	Not yet supported	Not yet supported
data-driven styling	Not yet supported	Not yet supported	Not yet supported

circle-pitch-scale

Optional Enum *One of* map, viewport. *Defaults to* map.

Controls the scaling behavior of the circle when the map is pitched.

map Circles are scaled according to their apparent distance to the camera.

viewport Circles are not scaled.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.21.0	Not yet supported	Not yet supported
data-driven styling	Not yet supported	Not yet supported	Not yet supported

circle-stroke-width

Optional Number. *Units in* pixels. *Defaults to* 5.

The width of the circle's stroke. Strokes are placed outside of the `circle-radius`.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.29.0	>= 17.1	>= 2.10.0
data-driven styling	>= 0.29.0	>= 17.1	>= 2.10.0

circle-stroke-color

Optional Color. *Defaults to* #000000.

The stroke color of the circle.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.29.0	>= 17.1	>= 2.4.0
data-driven styling	>= 0.29.0	>= 17.1	>= 2.4.0

circle-stroke-opacity

Optional Number. *Defaults to* 1.

The opacity of the circle's stroke.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.29.0	>= 17.1	Not yet supported
data-driven styling	>= 0.29.0	>= 17.1	Not yet supported

fill-extrusion

Layout Properties

visibility

Optional Enum. *One of* visible, none. *Defaults to* visible.

Whether this layer is displayed.

visible The layer is shown.

none The layer is not shown.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.27.0	>= 17.1	Not yet supported

Paint Properties

fill-extrusion-opacity

Optional Number. *Defaults to* 1.

The opacity of the entire fill extrusion layer. This is rendered on a per-layer, not per-feature, basis, and data-driven styling is not available.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.27.0	>= 17.1	Not yet supported

fill-extrusion-color

Optional Color. *Defaults to* #000000. *Disabled by* fill-extrusion-pattern.

The base color of the extruded fill. The extrusion's surfaces will be shaded differently based on this color in combination with the root `light` settings. If this color is specified as `rgba` with an alpha component, the alpha component will be ignored; use `fill-extrusion-opacity` to set layer opacity.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.27.0	>= 17.1	Not yet supported

fill-extrusion-translate

Optional Array. *Units in* pixels. *Defaults to* 0,0.

The geometry's offset. Values are [x, y] where negatives indicate left and up (on the flat plane), respectively.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.27.0	Not yet supported	Not yet supported
data-driven styling	Not yet supported	Not yet supported	Not yet supported

Optional Enum *One of* map, viewport. *Defaults to* map. *Requires* fill-extrusion-translate.

Controls the translation reference point.

map The fill extrusion is translated relative to the map.

viewport The fill extrusion is translated relative to the viewport.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.27.0	Not yet supported	Not yet supported
data-driven styling	Not yet supported	Not yet supported	Not yet supported

fill-extrusion-pattern

Optional String.

Name of image in sprite to use for drawing images on extruded fills. For seamless patterns, image width and height must be a factor of two (2, 4, 8, . . . , 512).

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.27.0	Not yet supported	Not yet supported
data-driven styling	Not yet supported	Not yet supported	Not yet supported

fill-extrusion-height

Optional Number *Units in* meters. *Defaults to* 0.

The height with which to extrude this layer.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.27.0	Not yet supported	Not yet supported
data-driven styling	>= 0.27.0	Not yet supported	Not yet supported

fill-extrusion-base

Optional *Number* *Units in* meters. *Defaults to* 0. *Requires* fill-extrusion-height.

The height with which to extrude the base of this layer. Must be less than or equal to `fill-extrusion-height`.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.27.0	Not yet supported	Not yet supported
data-driven styling	>= 0.27.0	Not yet supported	Not yet supported

Types

A Mapbox style contains values of various types, most commonly as values for the style properties of a layer.

Color

Colors are written as JSON strings in a variety of permitted formats: HTML-style hex values, rgb, rgba, hsl, and hsla. Predefined HTML colors names, like `yellow` and `blue`, are also permitted.

```
{
  "line-color": "#ff0",
  "line-color": "#ffff00",
  "line-color": "rgb(255, 255, 0)",
  "line-color": "rgba(255, 255, 0, 1)",
  "line-color": "hsl(100, 50%, 50%)",
  "line-color": "hsla(100, 50%, 50%, 1)",
  "line-color": "yellow"
}
```

Especially of note is the support for hsl, which can be easier to reason about than rgb().

Enum

One of a fixed list of string values. Use quotes around values.

```
{
  "text-transform": "uppercase"
}
```

String

A string is basically just text. In Mapbox styles, you're going to put it in quotes. Strings can be anything, though pay attention to the case

of `text-field` - it actually will refer to features, which you refer to by putting them in curly braces, as seen in the example below.

```
{
  "text-field": "{MY_FIELD}"
}
```

Boolean

Boolean means yes or no, so it accepts the values `true` or `false`.

```
{
  "fill-enabled": true
}
```

Number

A number value, often an integer or floating point (decimal number). Written without quotes.

```
{
  "text-size": 24
}
```

Array

Arrays are comma-separated lists of one or more numbers in a specific order. For example, they're used in line dash arrays, in which the numbers specify intervals of line, break, and line again.

```
{
  "line-dasharray": [2, 4]
}
```

Expressions

The value for any layout property, paint property, or filter may be specified as an expression. An expression defines a formula for computing the value of the property using the operators described below. The set of expression operators provided by Mapbox GL includes:

- Mathematical operators for performing arithmetic and other operations on numeric values
- Logical operators for manipulating boolean values and making conditional decisions
- String operators for manipulating strings
- Data operators, providing access to the properties of source features

- Camera operators, providing access to the parameters defining the current map view

Expressions are represented as JSON arrays. The first element of an expression array is a string naming the expression operator, e.g. `"*"` or `"case"`. Subsequent elements (if any) are the arguments to the expression. Each argument is either a literal value (a string, number, boolean, or null), or another expression array.

[expression_name, argument_0, argument_1, . . .]

Data expressions

A *data expression* is any expression that access feature data – that is, any expression that uses one of the data operators: `get`, `has`, `id`, `geometry-type`, `properties`, or `feature-state`. Data expressions allow a feature's properties or state to determine its appearance. They can be used to differentiate features within the same layer and to create data visualizations.

```
{
    "circle-color": [
        "rgb",
        // red is␣
↪higher when feature.properties.temperature is higher
        ["get", "temperature"],
        // green is always zero
        0,
        // blue is␣
↪higher when feature.properties.temperature is lower
        ["-", 100, ["get", "temperature"]]
    ]
}
```

This example uses the `get` operator to obtain the temperature value of each feature. That value is used to compute arguments to the `rgb` operator, defining a color in terms of its red, green, and blue components.

Data expressions are allowed as the value of the `filter` property, and as values for most paint and layout properties. However, some paint and layout properties do not yet support data expressions. The level of support is indicated by the "data-driven styling" row of the "SDK Support" table for each property. Data expressions with the `feature-state` operator are allowed only on paint properties.

Camera expressions

A *camera expression* is any expression that uses the `zoom operator`. Such expressions allow the the appearance of a layer to change with the map's zoom level. Camera expressions can be used to create the appearance of depth and to control data density.

```
{
    "circle-radius": [
        "interpolate", ["linear"], ["zoom"],
        ␣
↪  // zoom is 5 (or less) -> circle radius will be 1px
        5, 1,
```

```
            //
↪ zoom is 10 (or greater) -> circle radius will be 5px
            10, 5
        ]
}
```

This example uses the `interpolate` operator to define a linear relationship between zoom level and circle size using a set of input-output pairs. In this case, the expression indicates that the circle radius should be 1 pixel when the zoom level is 5 or below, and 5 pixels when the zoom is 10 or above. In between, the radius will be linearly interpolated between 1 and 5 pixels

Camera expressions are allowed anywhere an expression may be used. However, when a camera expression used as the value of a layout or paint property, it must be in one of the following forms:

```
[ "interpolate", interpolation, ["zoom"], ... ]
```

Or:

```
[ "step", ["zoom"], ... ]
```

Or:

```
[
    "let",
    ... variable bindings...,
    [ "interpolate", interpolation, ["zoom"], ... ]
]
```

Or:

```
[
    "let",
    ... variable bindings...,
    [ "step", ["zoom"], ... ]
]
```

That is, in layout or paint properties, `["zoom"]` may appear only as the input to an outer `interpolate` or `step` expression, or such an expression within a `let` expression.

There is an important difference between layout and paint properties in the timing of camera expression evaluation. Paint property camera expressions are re-evaluated whenever the zoom level changes, even fractionally. For example, a paint property camera expression will be re-evaluated continuously as the map moves between zoom levels 4.1 and 4.6. On the other hand, a layout property camera expression is evaluated only at integer zoom levels. It will not be re-evaluated as the zoom changes from 4.1 to 4.6 – only if it goes above 5 or below 4.

Composition

A single expression may use a mix of data operators, camera operators, and other operators. Such composite expressions allows a

layer's appearance to be determined by a combination of the zoom level and individual feature properties.

```
{
    "circle-radius": [
        "interpolate", ["linear"], ["zoom"],
        // when zoom is 0, set each feature's␣
↪circle radius to the value of its "rating" property
        0, ["get", "rating"],

↪// when zoom is 10, set each feature's circle radius␣
↪to four times the value of its "rating" property
        10, ["*", 4, ["get", "rating"]]
    ]
}
```

An expression that uses both data and camera operators is considered both a data expression and a camera expression, and must adhere to the restrictions described above for both.

Type system

The input arguments to expressions, and their result values, use the same set of types as the rest of the style specification: boolean, string, number, color, and arrays of these types. Furthermore, expressions are type safe: each use of an expression has a known result type and required argument types, and the SDKs verify that the result type of an expression is appropriate for the context in which it is used. For example, the result type of an expression in the `filter` property must be boolean, and the arguments to the + operator must be numbers.

When working with feature data, the type of a feature property value is typically not known ahead of time by the SDK. In order to preserve type safety, when evaluating a data expression, the SDK will check that the property value is appropriate for the context. For example, if you use the expression `["get", "feature-color"]` for the `circle-color` property, the SDK will verify that the `feature-color` value of each feature is a string identifying a valid color. If this check fails, an error will be indicated in an SDK-specific way (typically a log message), and the default value for the property will be used instead.

In most cases, this verification will occur automatically wherever it is needed. However, in certain situations, the SDK may be unable to automatically determine the expected result type of a data expression from surrounding context. For example, it is not clear whether the expression `["<", ["get", "a"], ["get", "b"]]` is attempting to compare strings or numbers. In situations like this, you can use one of the type assertion expression operators to indicate the expected type of a data expression: `["<", ["number", ["get", "a"]], ["number", ["get", "b"]]]`. A type assertion checks that the feature data actually matches the expected type of the data expression. If this check fails, it produces an error and causes the whole expression to fall back to the default value for the property being defined. The assertion operators are `array`, `boolean`, `number`, and `string`.

Expressions perform only one kind of implicit type conversion: a data expression used in a context where a color is expected will convert a string representation of a color to a color value. In all other cases, if you want to convert between types, you must use one of the type conversion expression operators: `to-boolean`, `to-number`, `to-string`, or `to-color`. For example, if you have a feature property that stores numeric values in string format, and you want to use those values as numbers rather than strings, you can use an expression such as `["to-number", ["get", "property-name"]]`.

Expression reference

Types

The expressions in this section are provided for the purpose of testing for and converting between different data types like strings, numbers, and boolean values.

Often, such tests and conversions are unnecessary, but they may be necessary in some expressions where the type of a certain subexpression is ambiguous. They can also be useful in cases where your feature data has inconsistent types; for example, you could use to-number to make sure that values like `"1.5"` (instead of `1.5`) are treated as numeric values.

array

Asserts that the input is an array (optionally with a specific item type and length). If, when the input expression is evaluated, it is not of the asserted type, then this assertion will cause the whole expression to be aborted.

```
["array", value]: array
```

```
["array", type:␣
↪"string" | "number" | "boolean", value]: array<type>
```

```
["array",
    type: "string" | "number" | "boolean",
    N: number (literal),
    value
]: array<type, N>
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.41.0	>= 20.0	>= 3.0.0

boolean

Asserts that the input value is a boolean. If multiple values are provided, each one is evaluated in order until a boolean is obtained. If

none of the inputs are booleans, the expression is an error.

```
["boolean", value]: boolean
```

```
["boolean", value,
↪ fallback: value, fallback: value, ...]: boolean
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.41.0	>= 20.0	>= 3.0.0

collator

Returns a `collator` for use in locale-dependent comparison operations. The `case-sensitive` and `diacritic-sensitive` options default to `false`. The locale argument specifies the IETF language tag of the locale to use. If none is provided, the default locale is used. If the requested locale is not available, the `collator` will use a system-defined fallback locale. Use `resolved-locale` to test the results of locale fallback behavior.

```
["collator",
    { "case-sensitive": boolean,
↪ "diacritic-sensitive": boolean, "locale": string }
]: collator
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.45.0	>= Not yet supported	>= Not yet supported

format

Returns `formatted` text containing annotations for use in mixed-format `text-field` entries. If set, the `text-font` argument overrides the font specified by the root layout properties. If set, the `font-scale` argument specifies a scaling factor relative to the `text-size` specified in the root layout properties.

```
["format",
    input_1: string, options_1:␣
↪{ "font-scale": number, "text-font": array<string> },
    ...,
    input_n: string, options_n:␣
↪{ "font-scale": number, "text-font": array<string> }
]: formatted
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.48.0	>= Not yet supported	>= Not yet supported

literal

Provides a literal array or object value.

```
["literal", [...] (JSON array literal)]: array<T, N>
```

```
["literal", {...} (JSON object literal)]: Object
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.41.0	>= 20.0	>= 3.0.0

number

Asserts that the input value is a number. If multiple values are provided, each one is evaluated in order until a number is obtained. If none of the inputs are numbers, the expression is an error.

```
["number", value]: number
```

```
["number",␣
↪value, fallback: value, fallback: value, ...]: number
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.41.0	>= 20.0	>= 3.0.0

object

Asserts that the input value is an object. If multiple values are provided, each one is evaluated in order until an object is obtained. If none of the inputs are objects, the expression is an error.

```
["object", value]: object
```

```
["object",␣
↪value, fallback: value, fallback: value, ...]: object
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.41.0	>= 20.0	>= 3.0.0

string

Asserts that the input value is a string. If multiple values are provided, each one is evaluated in order until a string is obtained. If none of the inputs are strings, the expression is an error.

```
["string", value]: string
```

```
["string",␣
↪value, fallback: value, fallback: value, ...]: string
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.41.0	>= 20.0	>= 3.0.0

to-boolean

Converts the input value to a boolean. The result is false when then input is an empty string, 0, `false`, `null`, or `NaN`; otherwise it is `true`.

```
["to-boolean", value]: boolean
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.41.0	>= 20.0	>= 3.0.0

to-color

Converts the input value to a color. If multiple values are provided, each one is evaluated in order until the first successful conversion is obtained. If none of the inputs can be converted, the expression is an error.

```
["to-color",
↪ value, fallback: value, fallback: value, ...]: color
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.41.0	>= 20.0	>= 3.0.0

to-number

Converts the input value to a number, if possible. If the input is `null` or `false`, the result is 0. If the input is `true`, the result is 1. If the input is a string, it is converted to a number as specified by the "ToNumber Applied to the String Type" algorithm of the ECMAScript Language Specification. If multiple values are provided, each one is evaluated in order until the first successful conversion is obtained. If none of the inputs can be converted, the expression is an error.

```
["to-number",␣
↪value, fallback: value, fallback: value, ...]: number
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.41.0	>= 20.0	>= 3.0.0

to-string

Converts the input value to a string. If the input is `null`, the result is `""`. If the input is a boolean, the result is `"true"` or `"false"`. If the input is a number, it is converted to a string as specified by the "NumberToString" algorithm of the ECMAScript Language Specification. If the input is a color, it is converted to a string of the form `"rgba(r,g,b,a)"`, where `r`, `g`, and `b` are numerals ranging from 0 to 255, and `a` ranges from 0 to 1. Otherwise, the input is converted to a string in the format specified by the `JSON.stringify` function of the ECMAScript Language Specification.

```
["to-string", value]: string
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.41.0	>= 20.0	>= 3.0.0

typeof

Returns a string describing the type of the given value.

```
["typeof", value]: string
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.41.0	>= 20.0	>= 3.0.0

Feature data

feature-state

Retrieves a property value from the current feature's state. Returns null if the requested property is not present on the feature's state. A feature's state is not part of the GeoJSON or vector tile data, and must be set programmatically on each feature. Note that `["feature-state"]` can only be used with paint properties that support data-driven styling.

```
["feature-state", string]: value
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.46.0	>= Not yet supported	>= Not yet supported

geometry-type

Gets the feature's geometry type: Point, MultiPoint, LineString, MultiLineString, Polygon, MultiPolygon.

```
["geometry-type"]: string
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.41.0	>= 20.0	>= 3.0.0

id

Gets the feature's id, if it has one.

```
["id"]: value
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.41.0	>= 20.0	>= 3.0.0

line-progress

Gets the progress along a gradient line. Can only be used in the line-gradient property.

```
["line-progress"]: number
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.45.0	>= Not yet supported	>= Not yet supported

properties

Gets the feature properties object. Note that in some cases, it may be more efficient to use ["get", "property_name"] directly.

```
["properties"]: object
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.41.0	>= Not yet supported	>= 3.0.0

Lookup

at

Retrieves an item from an array.

```
["at", number, array]: ItemType
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.41.0	>= 20.0	>= 3.0.0

get

Retrieves a property value from the current feature's properties, or from another object if a second argument is provided. Returns null if the requested property is missing.

```
["get", string]: value
```

```
["get", string, object]: value
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.41.0	>= 20.0	>= 3.0.0

has

Tests for the presence of an property value in the current feature's properties, or from another object if a second argument is provided.

```
["has", string]: boolean
```

```
["has", string, object]: boolean
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.41.0	>= 20.0	>= 3.0.0

length

Gets the length of an array or string.

```
["length", string | array | value]: number
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.41.0	>= 20.0	>= 3.0.0

Decision

The expressions in this section can be used to add conditional logic to your styles. For example, the `'case'` expression provides basic "if/then/else" logic, and `'match'` allows you to map specific values of an input expression to different output expressions.

!

Logical negation. Returns true if the input is false, and false if the input is true.

```
["!", boolean]: boolean
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.41.0	>= 20.0	>= 3.0.0

!=

Returns `true` if the input values are not equal, `false` otherwise. The comparison is strictly typed: values of different runtime types are always considered unequal. Cases where the types are known to be different at parse time are considered invalid and will produce a parse error. Accepts an optional `collator` argument to control locale-dependent string comparisons.

```
["!=", value, value]: boolean
```

```
["!=", value, value, collator]: boolean
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.41.0	>= 20.0	>= 3.0.0
collator	>= 0.45.0	>= Not yet supported	>= Not yet supported

<

Returns `true` if the first input is strictly less than the second, `false` otherwise. The arguments are required to be either both strings or both numbers; if during evaluation they are not, expression evaluation produces an error. Cases where this constraint is known not to hold at parse time are considered in valid and will produce a parse error. Accepts an optional `collator` argument to control locale-dependent string comparisons.

```
["<", value, value]: boolean
```

```
["<", value, value, collator]: boolean
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.41.0	>= 20.0	>= 3.0.0
collator	>= 0.45.0	>= Not yet supported	>= Not yet supported

<=

Returns `true` if the first input is less than or equal to the second, `false` otherwise. The arguments are required to be either both strings or both numbers; if during evaluation they are not, expression evaluation produces an error. Cases where this constraint is known not to hold at parse time are considered in valid and will produce a parse error. Accepts an optional `collator` argument to control locale-dependent string comparisons.

```
["<=", value, value]: boolean
```

```
["<=", value, value, collator]: boolean
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.41.0	>= 20.0	>= 3.0.0
collator	>= 0.45.0	>= Not yet supported	>= Not yet supported

==

Returns `true` if the input values are equal, `false` otherwise. The comparison is strictly typed: values of different runtime types are always considered unequal. Cases where the types are known to be different at parse time are considered invalid and will produce a parse error. Accepts an optional `collator` argument to control locale-dependent string comparisons.

```
["==", value, value]: boolean
```

```
["==", value, value, collator]: boolean
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.41.0	>= 20.0	>= 3.0.0
collator	>= 0.45.0	>= Not yet supported	>= Not yet supported

>

Returns `true` if the first input is strictly greater than the second, `false` otherwise. The arguments are required to be either both

strings or both numbers; if during evaluation they are not, expression evaluation produces an error. Cases where this constraint is known not to hold at parse time are considered in valid and will produce a parse error. Accepts an optional `collator` argument to control locale-dependent string comparisons.

```
[">", value, value]: boolean
```

```
[">", value, value, collator]: boolean
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.41.0	>= 20.0	>= 3.0.0
collator	>= 0.45.0	>= Not yet supported	>= Not yet supported

>=

Returns `true` if the first input is greater than or equal to the second, `false` otherwise. The arguments are required to be either both strings or both numbers; if during evaluation they are not, expression evaluation produces an error. Cases where this constraint is known not to hold at parse time are considered in valid and will produce a parse error. Accepts an optional `collator` argument to control locale-dependent string comparisons.

```
[">=", value, value]: boolean
```

```
[">=", value, value, collator]: boolean
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.41.0	>= 20.0	>= 3.0.0
collator	>= 0.45.0	>= Not yet supported	>= Not yet supported

all

Returns `true` if all the inputs are true, `false` otherwise. The inputs are evaluated in order, and evaluation is short-circuiting: once an input expression evaluates to `false`, the result is false and no further input expressions are evaluated.

```
["all", boolean, boolean]: boolean
```

```
["all", boolean, boolean, ...]: boolean
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.41.0	>= 20.0	>= 3.0.0

any

Returns `true` if any of the inputs are `true`, `false` otherwise. The inputs are evaluated in order, and evaluation is short-circuiting: once an input expression evaluates to `true`, the result is true and no further input expressions are evaluated.

```
["any", boolean, boolean]: boolean
```

```
["any", boolean, boolean, ...]: boolean
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.41.0	>= 20.0	>= 3.0.0

case

Selects the first output whose corresponding test condition evaluates to true.

```
["case",
    condition: boolean, output: OutputType,
↪ condition: boolean, output: OutputType, ...,
    default: OutputType
]: OutputType
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.41.0	>= 20.0	>= 3.0.0

coalesce

Evaluates each expression in turn until the first non-null value is obtained, and returns that value.

```
["coalesce", OutputType, OutputType, ...]: OutputType
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.41.0	>= 20.0	>= 3.0.0

match

Selects the output whose label value matches the input value, or the fallback value if no match is found. The input can be any expression (e.g. `["get", "building_type"]`). Each label must either be a single literal value or an array of literal values (e.g. `"a"` or `["c", "b"]`), and those values must be all strings or all numbers. (The values `"1"` and `1` cannot both be labels in the same match expression.)

Each label must be unique. If the input type does not match the type of the labels, the result will be the fallback value.

```
["match",
    input: InputType (number or string),
    label_1: InputType␣
↪| [InputType, InputType, ...], output_1: OutputType,
    label_n: InputType | [InputType,
↪ InputType, ...], output_n: OutputType, ...,
    default: OutputType
]: OutputType
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.41.0	>= 20.0	>= 3.0.0

Ramps, scales, curves

interpolate

Produces continuous, smooth results by interpolating between pairs of input and output values ("stops"). The `input` may be any numeric expression (e.g., `["get", "population"]`). Stop inputs must be numeric literals in strictly ascending order. The output type must be `number`, `array<number>`, or `color`.

Interpolation types:

- `["linear"]`: interpolates linearly between the pair of stops just less than and just greater than the input.
- `["exponential", base]`: interpolates exponentially between the stops just less than and just greater than the input. base controls the rate at which the output increases: higher values make the output increase more towards the high end of the range. With values close to 1 the output increases linearly.
- `["cubic-bezier", x1, y1, x2, y2]`: interpolates using the cubic bezier curve defined by the given control points.

```
["interpolate",
    interpolation: ["linear"] | ["exponential
↪", base] | ["cubic-bezier", x1, y1, x2, y2 ],
    input: number,
    stop_input_1: number, stop_output_1: OutputType,
  ␣
↪ stop_input_n: number, stop_output_n: OutputType, ...
]: OutputType (number, array<number>, or Color)
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.42.0	>= Not yet supported	>= 3.0.0

interpolate-hcl

Produces continuous, smooth results by interpolating between pairs of input and output values ("stops"). Works like `interpolate`, but the output type must be `color`, and the interpolation is performed in the Hue-Chroma-Luminance color space.

```
["interpolate-hcl",
    interpolation: ["linear"] | ["exponential
↪", base] | ["cubic-bezier", x1, y1, x2, y2 ],
    input: number,
    stop_input_1: number, stop_output_1: Color,
    stop_input_n: number, stop_output_n: Color, ...
]: Color
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.49.0	>= Not yet supported	>= 3.0.0

interpolate-lab

Produces continuous, smooth results by interpolating between pairs of input and output values ("stops"). Works like `interpolate`, but the output type must be `color`, and the interpolation is performed in the CIELAB color space.

```
["interpolate-lab",
    interpolation: ["linear"] | ["exponential
↪", base] | ["cubic-bezier", x1, y1, x2, y2 ],
    input: number,
    stop_input_1: number, stop_output_1: Color,
    stop_input_n: number, stop_output_n: Color, ...
]: Color
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.49.0	>= Not yet supported	>= 3.0.0

step

Produces discrete, stepped results by evaluating a piecewise-constant function defined by pairs of input and output values ("stops"). The `input` may be any numeric expression (e.g., `["get", "population"]`). Stop inputs must be numeric literals in strictly ascending order. Returns the output value of the stop just less than the input, or the first input if the input is less than the first stop.

```
["step",
    input: number,
    stop_output_0: OutputType,
```

```
    stop_input_1: number, stop_output_1: OutputType,
␣
↪ stop_input_n: number, stop_output_n: OutputType, ...
]: OutputType
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.42.0	>= Not yet supported	>= 3.0.0

Variable binding

let

Binds expressions to named variables, which can then be referenced in the result expression using ["var", "variable_name"].

```
["let",
    string (alphanumeric literal),
↪ any, string (alphanumeric literal), any, ...,
    OutputType
]: OutputType
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.41.0	>= Not yet supported	>= 3.0.0

var

References variable bound using "let".

```
["var", previously bound␣
↪variable name]: the type of the bound expression
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.41.0	>= Not yet supported	>= 3.0.0

String

concat

Returns a string consisting of the concatenation of the inputs. Each input is converted to a string as if by `to-string`.

```
["concat", value, value, ...]: string
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.41.0	>= 20.0	>= 3.0.0

downcase

Returns the input string converted to lowercase. Follows the Unicode Default Case Conversion algorithm and the locale-insensitive case mappings in the Unicode Character Database.

```
["downcase", string]: string
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.41.0	>= 20.0	>= Not yet supported

is-supported-script

Returns `true` if the input string is expected to render legibly. Returns `false` if the input string contains sections that cannot be rendered without potential loss of meaning (e.g. Indic scripts that require complex text shaping, or right-to-left scripts if the the `mapbox-gl-rtl-text` plugin is not in use in Mapbox GL JS).

```
["is-supported-script", string]: boolean
```

resolved-locale

Returns the IETF language tag of the locale being used by the provided `collator`. This can be used to determine the default system locale, or to determine if a requested locale was successfully loaded.

```
["resolved-locale", collator]: string
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.45.0	>= Not yet supported	>= Not yet supported

upcase

Returns the input string converted to uppercase. Follows the Unicode Default Case Conversion algorithm and the locale-insensitive case mappings in the Unicode Character Database.

```
["upcase", string]: string
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.41.0	>= 20.0	>= Not yet supported

Color

rgb

Creates a color value from red, green, and blue components, which must range between 0 and 255, and an alpha component of 1. If any component is out of range, the expression is an error.

```
["rgb", number, number, number]: color
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.41.0	>= 20.0	>= 3.0.0

rgba

Creates a color value from red, green, blue components, which must range between 0 and 255, and an alpha component which must range between 0 and 1. If any component is out of range, the expression is an error.

```
["rgba", number, number, number, number]: color
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.41.0	>= Not yet supported	>= 3.0.0

to-rgba

Returns a four-element array containing the input color's red, green, blue, and alpha components, in that order.

```
["to-rgba", color]: array<number, 4>
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.41.0	>= Not yet supported	>= 3.0.0

Math

-

For two inputs, returns the result of subtracting the second input from the first. For a single input, returns the result of subtracting it from 0.

```
["-", number, number]: number
```

```
["-", number]: number
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.41.0	>= 20.0	>= 3.0.0

*

Returns the product of the inputs.

```
["*", number, number, ...]: number
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.41.0	>= 20.0	>= 3.0.0

/

Returns the result of floating point division of the first input by the second.

```
["/", number, number]: number
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.41.0	>= 20.0	>= 3.0.0

%

Returns the remainder after integer division of the first input by the second.

```
["%", number, number]: number
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.41.0	>= 20.0	>= 3.0.0

^

Returns the result of raising the first input to the power specified by the second.

```
["^", number, number]: number
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.41.0	>= 20.0	>= 3.0.0

+

Returns the sum of the inputs.

```
["+", number, number, ...]: number
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.41.0	>= 20.0	>= 3.0.0

abs

Returns the absolute value of the input.

```
["abs", number]: number
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.45.0	>= Not yet supported	>= 3.0.0

acos

Returns the arccosine of the input.

```
["acos", number]: number
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.41.0	>= 20.0	>= 3.0.0

asin

Returns the arcsine of the input.

```
["asin", number]: number
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.41.0	>= 20.0	>= 3.0.0

atan

Returns the arctangent of the input.

```
["atan", number]: number
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.41.0	>= 20.0	>= 3.0.0

ceil

Returns the smallest integer that is greater than or equal to the input.

```
["ceil", number]: number
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.45.0	>= Not yet supported	>= 3.0.0

cos

Returns the cosine of the input.

```
["cos", number]: number
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.41.0	>= 20.0	>= 3.0.0

e

Returns the mathematical constant e.

```
["e"]: number
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.41.0	>= 20.0	>= 3.0.0

floor

Returns the largest integer that is less than or equal to the input.

```
["floor", number]: number
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.45.0	>= Not yet supported	>= 3.0.0

ln

Returns the natural logarithm of the input.

```
["ln", number]: number
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.41.0	>= 20.0	>= 3.0.0

ln2

Returns mathematical constant ln(2).

```
["ln2"]: number
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.41.0	>= 20.0	>= 3.0.0

log10

Returns the base-ten logarithm of the input.

```
["log10", number]: number
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.41.0	>= 20.0	>= 3.0.0

log2

Returns the base-two logarithm of the input.

```
["log2", number]: number
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.41.0	>= 20.0	>= 3.0.0

max

Returns the maximum value of the inputs.

```
["max", number, number, ...]: number
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.41.0	>= 20.0	>= 3.0.0

min

Returns the minimum value of the inputs.

```
["min", number, number, ...]: number
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.41.0	>= 20.0	>= 3.0.0

pi

Returns the mathematical constant pi.

```
["pi"]: number
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.41.0	>= 20.0	>= 3.0.0

round

Rounds the input to the nearest integer. Halfway values are rounded away from zero. For example, `["round", -1.5]` evaluates to -2.

```
["round", number]: number
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.45.0	>= Not yet supported	>= 3.0.0

sin

Returns the sine of the input.

```
["sin", number]: number
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.41.0	>= 20.0	>= 3.0.0

sqrt

Returns the square root of the input.

```
["sqrt", number]: number
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.42.0	>= 20.0	>= 3.0.0

tan

Returns the tangent of the input.

```
["tan", number]: number
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.41.0	>= 20.0	>= 3.0.0

Zoom

zoom

Gets the current zoom level. Note that in style layout and paint properties, ["zoom"] may only appear as the input to a top-level "step" or "interpolate" expression.

```
["zoom"]: number
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.41.0	>= 20.0	>= 3.0.0

Heatmap

heatmap-density

Gets the kernel density estimation of a pixel in a heatmap layer, which is a relative measure of how many data points are crowded around a particular pixel. Can only be used in the `heatmap-color` property.

```
["heatmap-density"]: number
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.41.0	>= Not yet supported	>= Not yet supported

Other

Function

The value for any layout or paint property may be specified as a *function*. Functions allow you to make the appearance of a map feature change with the current zoom level and/or the feature's properties.

stops

Required (except for identity functions) :ref:'types-array'.

Functions are defined in terms of input and output values. A set of one input value and one output value is known as a "stop."

property

Optional *String*.

If specified, the function will take the specified feature property as an input. See *Zoom Functions and Property Functions* for more information.

base

Optional *Number*. *Default is* 1.

The exponential base of the interpolation curve. It controls the rate at which the function output increases. Higher values make the output increase more towards the high end of the range. With values close to 1 the output increases linearly.

type

Optional Enum. One of identity, exponential, interval, categorical.

identity functions return their input as their output.

exponential functions generate an output by interpolating between stops just less than and just greater than the function input. The domain must be numeric. This is the default for properties marked with , the "exponential" symbol.

interval functions return the output value of the stop just less than the function input. The domain must be numeric. This is the default for properties marked with , the "interval" symbol.

categorical functions return the output value of the stop equal to the function input.

default

A value to serve as a fallback function result when a value isn't otherwise available. It is used in the following circumstances:

- In categorical functions, when the feature value does not match any of the stop domain values.
- In property and zoom-and-property functions, when a feature does not contain a value for the specified property.
- In identity functions, when the feature value is not valid for the style property (for example, if the function is being used for a circle-color property but the feature property value is not a string or not a valid color).
- In interval or exponential property and zoom-and-property functions, when the feature value is not numeric.

If no default is provided, the style property's default is used in these circumstances.

colorSpace

Optional Enum. One of rgb, lab, hcl.

The color space in which colors interpolated. Interpolating colors in perceptual color spaces like LAB and HCL tend to produce color ramps that look more consistent and produce colors that can be differentiated more easily than those interpolated in RGB space.

rgb Use the RGB color space to interpolate color values

lab Use the LAB color space to interpolate color values.

hcl Use the HCL color space to interpolate color values, interpolating the Hue, Chroma, and Luminance channels individually.

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	>= 17.1	>= 2.4.0
`property`	>= 0.18.0	>= 17.1	>= 2.4.0
`type`	>= 0.18.0	>= 17.1	>= 2.4.0
`exponential` type	>= 0.18.0	>= 17.1	>= 2.4.0
`interval` type	>= 0.18.0	>= 17.1	>= 2.4.0
`categorical` type	>= 0.18.0	>= 17.1	>= 2.4.0
`identity` type	>= 0.18.0	>= 17.1	>= 2.4.0
`default` type	>= 0.18.0	>= 17.1	>= 2.4.0
`colorSpace` type	>= 0.26.0	Not yet supported	>= 2.4.0

Zoom functions allow the appearance of a map feature to change with map's zoom level. Zoom functions can be used to create the illusion of depth and control data density. Each stop is an array with two elements: the first is a zoom level and the second is a function output value.

```
{
  "circle-radius": {
    "stops": [

      // zoom is 5 -> circle radius will be 1px
      [5, 1],

      // zoom is 10 -> circle radius will be 2px
      [10, 2]

    ]
  }
}
```

The rendered values of *Color*, *Number*, and *Array* properties are intepolated between stops. *Enum*, *Boolean*, and *String* property values cannot be intepolated, so their rendered values only change at the specified stops.

There is an important difference between the way that zoom functions render for *layout* and *paint* properties. Paint properties are continuously re-evaluated whenever the zoom level changes, even fractionally. The rendered value of a paint property will change, for example, as the map moves between zoom levels `4.1` and `4.6`. Layout properties, on the other hand, are evaluated only once for each integer zoom level. To continue the prior example: the rendering of a layout property will *not* change between zoom levels `4.1` and `4.6`, no matter what stops are specified; but at zoom level `5`, the function will be re-evaluated according to the function, and the property's rendered value will change. (You can include fractional zoom levels in a layout property zoom function, and it will affect the generated values; but, still, the rendering will only change at integer zoom levels.)

Property functions allow the appearance of a map feature to change with its properties. Property functions can be used to visually differentiate types of features within the same layer or create data visualizations. Each stop is an array with two elements, the first is a property input value and the second is a function output value. Note that support for property functions is not available across all properties and platforms at this time.

```
{
  "circle-color": {
    "property": "temperature",
    "stops": [

    // "temperature" is 0   -> circle color will be blue
      [0, 'blue'],

    // "temperature" is 100 -> circle color will be red
      [100, 'red']

    ]
  }
}
```

Zoom-and-property functions allow the appearance of a map feature to change with both its properties *and* zoom. Each stop is an array with two elements, the first is an object with a property input value and a zoom, and the second is a function output value. Note that support for property functions is not yet complete.

```
{
  "circle-radius": {
    "property": "rating",
    "stops": [

      // zoom is 0 and "rating" is 0 -> circle radius will be 0px
      [{zoom: 0, value: 0}, 0],

      // zoom is 0 and "rating" is 5 -> circle radius will be 5px
      [{zoom: 0, value: 5}, 5],

      // zoom is 20 and "rating" is 0 -> circle radius will be 0px
      [{zoom: 20, value: 0}, 0],

      // zoom is 20 and "rating" is 5 -> circle radius will be 20px
      [{zoom: 20, value: 5}, 20]

    ]
  }
}
```

Filter

A filter selects specific features from a layer. A filter is an array of one of the following forms:

Existential Filters

`["has", key]` feature[key] exists

`["!has", key]` feature[key] does not exist

Comparison Filters

`["==", key, value]` equality: feature[key] = value

`["!=", key, value]` inequality: feature[key] ≠ value

`[">", key, value]` greater than: feature[key] > value

`[">=", key, value]` greater than or equal: feature[key] ≥ value

`["<", key, value]` less than: feature[key] < value

`["<=", key, value]` less than or equal: feature[key] ≤ value

Set Membership Filters

`["in", key, v0, ..., vn]` set inclusion: feature[key] ∈ {v0, ..., vn}

`["!in", key, v0, ..., vn]` set exclusion: feature[key] ∉ {v0, ..., vn}

Combining Filters

`["all", f0, ..., fn]` logical `AND`: f0 ∧ ... ∧ fn

`["any", f0, ..., fn]` logical `OR`: f0 ∨ ... ∨ fn

`["none", f0, ..., fn]` logical `NOR`: ¬f0 ∧ ... ∧ ¬fn

A key must be a string that identifies a feature property, or one of the following special keys:

- `"$type"`: the feature type. This key may be used with the `"=="`, `"!="`, `"in"`, and `"!in"` operators. Possible values are `"Point"`, `"LineString"`, and `"Polygon"`.
- `"$id"`: the feature identifier. This key may be used with the `"=="`, `"!="`, `"has"`, `"!has"`, `"in"`, and `"!in"` operators.

A value (and v0, ..., vn for set operators) must be a *String*, *Number*, or *Boolean* to compare the property value against.

Set membership filters are a compact and efficient way to test whether a field matches any of multiple values.

The comparison and set membership filters implement strictly-typed comparisons; for example, all of the following evaluate to false: `0 < "1"`, `2 == "2"`, `"true" in [true, false]`.

The `"all"`, `"any"`, and `"none"` filter operators are used to create compound filters. The values f0, ..., fn must be filter expressions themselves.

```
["==", "$type", "LineString"]
```

This filter requires that the `class` property of each feature is equal to either "street_major", "street_minor", or "street_limited".

```
["in", "class
↪", "street_major", "street_minor", "street_limited"]
```

The combining filter "all" takes the three other filters that follow it and requires all of them to be true for a feature to be included: a feature must have a `class` equal to "street_limited", its `admin_level` must be greater than or equal to 3, and its type cannot be Polygon. You could change the combining filter to "any" to allow features matching any of those criteria to be included - features that are Polygons, but have a different `class` value, and so on.

```
[
  "all",
  ["==", "class", "street_limited"],
  [">=", "admin_level", 3],
  ["!in", "$type", "Polygon"]
]
```

Support	Mapbox	GeoTools	OpenLayers
basic functionality	>= 0.10.0	>= 17.1	>= 2.4.0
`has/!has`	>= 0.19.0	>= 17.1	>= 2.4.0

6.6.5 MBStyle Cookbook

The MBStyle Cookbook is a collection of MBStyle "recipes" for creating various types of map styles. Wherever possible, each example is designed to show off a single MBStyle layer so that code can be copied from the examples and adapted when creating MBStyles of your own. While not an exhaustive reference like the *MBStyle reference* the MBStyle cookbook is designed to be a practical reference, showing common style templates that are easy to understand.

The MBStyle Cookbook is divided into four sections: the first three for each of the vector types (points, lines, and polygons) and the fourth section for rasters to come. Each example in every section contains a screenshot showing actual GeoServer WMS output, a snippet of the MBStyle code for reference, and a link to download the full MBStyle.

Each section uses data created especially for the MBStyle Cookbook, with shapefiles for vector data and GeoTIFFs for raster data.

Data Type	Shapefile
Point	`mbstyle_cookbook_point.zip`
Line	`mbstyle_cookbook_line.zip`
Polygon	`mbstyle_cookbook_polygon.zip`

Points

Points are seemingly the simplest type of shape, possessing only position and no other dimensions. MBStyle has a `circle` type that can be styled to represent a point.

Example points layer

The `points layer` used for the examples below contains name and population information for the major cities of a fictional country. For reference, the attribute table for the points in this layer is included below.

`fid` (Feature ID)	`name` (City name)	`pop` (Population)
point.1	Borfin	157860
point.2	Supox City	578231
point.3	Ruckis	98159
point.4	Thisland	34879
point.5	Synopolis	24567
point.6	San Glissando	76024
point.7	Detrania	205609

`Download the points shapefile`

Simple point

This example specifies points be styled as red circles with a diameter of 6 pixels.

Fig. 6.269: Simple point

Code

Download the "Simple point" MBStyle JSON

```
{
  "version": 8,
  "name": "point-circle-test",
  "layers": [
    {
      "id": "point",
      "type": "circle",
      "paint": {
        "circle-radius": 3,
        "circle-color": "#FF0000",
        "circle-pitch-scale": "map"
      }
    }
  ]
}
```

Details

There is one layer in this MBStyle, which is the simplest possible situation. The "version" must always be set to 8. Layers is required for any MBStyle as an array. "id" is required and is a unique name for that layer. For our examples we will be setting the "type" to "circle".

Simple point with stroke

This example adds a stroke (or border) around the *Simple point*, with the stroke colored black and given a thickness of 2 pixels.

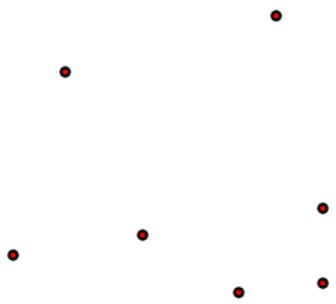


Fig. 6.270: Simple point with stroke

Code

Download the "Simple point with stroke" MBStyle JSON

```
{
  "version": 8,
  "name": "point-circle-test",
  "layers": [
    {
      "id": "point",
      "type": "circle",
      "paint": {
        "circle-radius": 3,
        "circle-color": "#FF0000",
        "circle-pitch-scale": "map",
        "circle-stroke-color": "#000000",
        "circle-stroke-width": 2
      }
    }
  ]
}
```

Details

This example is similar to the *Simple point* example. **Lines 12-13** specify the stroke, with **line 12** setting the color to black (`'#000000'`) and **line 13** setting the width to 2 pixels.

Lines

While lines can also seem to be simple shapes, having length but no width, there are many options and tricks for making lines display nicely.

Example lines layer

The `lines layer` used in the examples below contains road information for a fictional country. For reference, the attribute table for the points in this layer is included below.

fid (Feature ID)	name (Road name)	type (Road class)
line.1	Latway	highway
line.2	Crescent Avenue	secondary
line.3	Forest Avenue	secondary
line.4	Longway	highway
line.5	Saxer Avenue	secondary
line.6	Ridge Avenue	secondary
line.7	Holly Lane	local-road
line.8	Mulberry Street	local-road
line.9	Nathan Lane	local-road
line.10	Central Street	local-road
line.11	Lois Lane	local-road
line.12	Rocky Road	local-road
line.13	Fleet Street	local-road
line.14	Diane Court	local-road
line.15	Cedar Trail	local-road
line.16	Victory Road	local-road
line.17	Highland Road	local-road
line.18	Easy Street	local-road
line.19	Hill Street	local-road
line.20	Country Road	local-road
line.21	Main Street	local-road
line.22	Jani Lane	local-road
line.23	Shinbone Alley	local-road
line.24	State Street	local-road
line.25	River Road	local-road

Download the lines shapefile

Simple line

This example specifies lines be colored black with a thickness of 3 pixels.

Code

Download the "Simple line" MBStyle

```
{
  "version": 8,
  "name": "simple-line",
  "layers": [
    {
      "id": "simple-line",
      "type": "line",
      "paint": {
        "line-color": "#000000",
        "line-width": 3
      }
    }
```

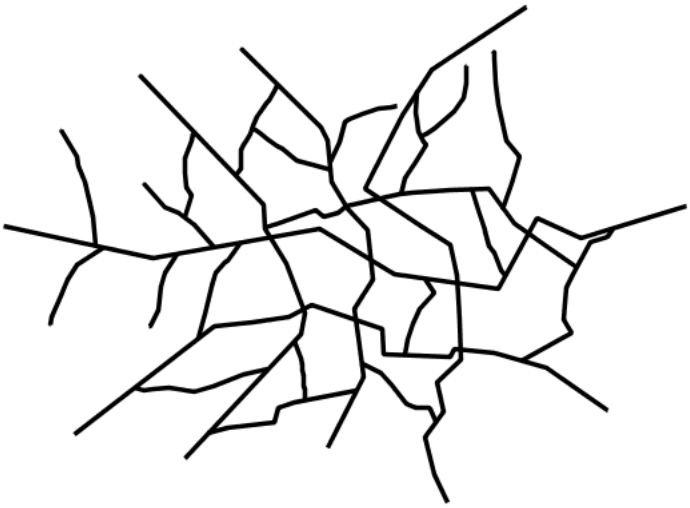


Fig. 6.271: Simple line

```
13                                      ]
14                                    }
```

Details

There is one layer style for this MBStyle, which is the simplest possible situation. Styling lines is accomplished using the line layer. **Line 9** specifies the color of the line to be black (`"#000000"`), while **line 10** specifies the width of the lines to be 3 pixels.

Line with border

This example shows how to draw lines with borders (sometimes called "cased lines"). In this case the lines are drawn with a 3 pixel blue center and a 1 pixel wide gray border.

Code

Download the "Line with border" MBStyle

```
1                                     {
2                                       "version": 8,
3                                       "name": "simple-borderedline",
4                                       "layers": [
5                                         {
6                                           "id": "simple-borderedline",
7                                           "type": "line",
8                                           "layout": {
```

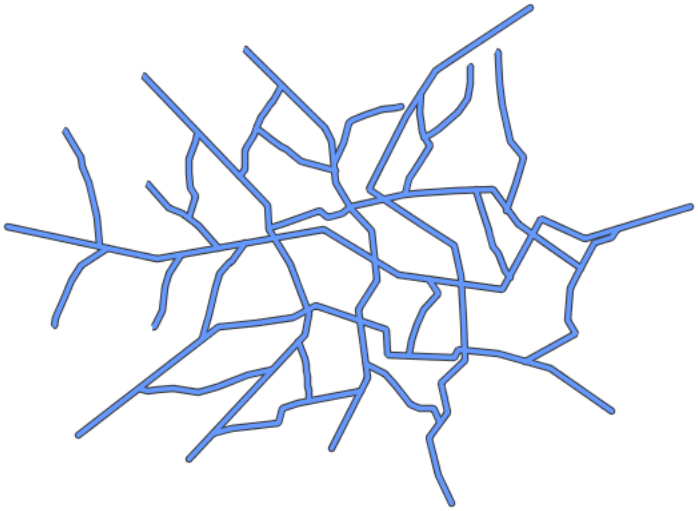


Fig. 6.272: Line with border

```
      "line-cap": "round"
    },
    "paint": {
      "line-color": "#333333",
      "line-width": 5
    }
  },
  {
    "id": "simple-line",
    "type": "line",
    "layout": {
      "line-cap": "round"
    },
    "paint": {
      "line-color": "#6699FF",
      "line-width": 3
    }
  }
 ]
}
```

Details

In this example we are drawing the lines twice to achieve the appearance of a line with a border. Since every line is drawn twice, the order of the rendering is *very* important. GeoServer renders `layers` in the order that they are presented in the MBStyle. In this style, the gray border lines are drawn first via the first layer style, followed by the blue center lines in a second layer style. This ensures that the blue lines are not obscured by the gray lines, and also ensures proper rendering at intersections, so that the blue lines "connect".

In this example, **lines 5-15** comprise the first layer style, which is the outer line (or "stroke"). **Line 12** specifies the color of the line to be dark gray (`"#333333"`), **line 13** specifies the width of this line to be 5 pixels, and in the `layout` **line 9** a `line-cap` parameter of `round` renders the ends of the line as rounded instead of flat. (When working with bordered lines using a round line cap ensures that the border connects properly at the ends of the lines.)

Lines 16-26 comprise the second `layer`, which is the the inner line (or "fill"). **Line 23** specifies the color of the line to be a medium blue (`"#6699FF"`), **line 24** specifies the width of this line to be 3 pixels, and in the `layout` **line 20** again renders the edges of the line to be rounded instead of flat.

The result is a 3 pixel blue line with a 1 pixel gray border, since the 5 pixel gray line will display 1 pixel on each side of the 3 pixel blue line.

Dashed line

This example alters the *Simple line* to create a dashed line consisting of 5 pixels of drawn line alternating with 2 pixels of blank space.

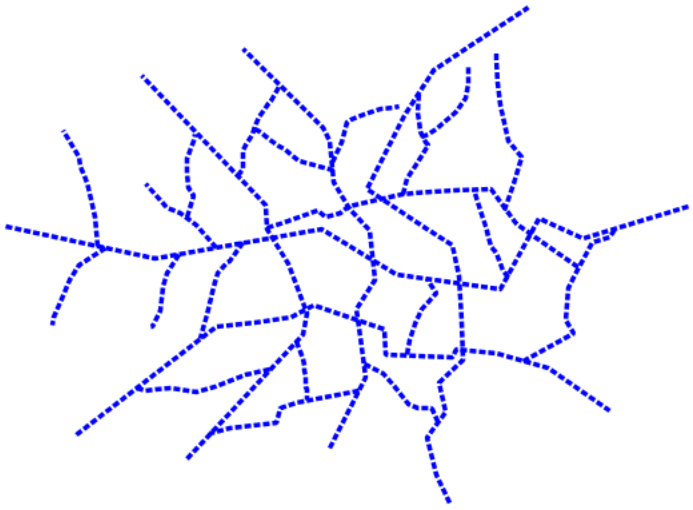


Fig. 6.273: Dashed line

Code

Download the "Dashed line" MBStyle

```
1  {
2    "version": 8,
3    "name": "simple-dashedline",
```

```
    "layers": [
      {
        "id": "simple-dashedline",
        "type": "line",
        "paint": {
          "line-color": "#0000FF",
          "line-width": 3,
          "line-dasharray": [5, 2]
        }
      }
    ]
}
```

Details

In this example, **line 9** sets the color of the lines to be blue (`"#0000FF"`) and **line 10** sets the width of the lines to be 3 pixels. **Line 11** determines the composition of the line dashes. The value of `[5, 2]` creates a repeating pattern of 5 pixels of drawn line, followed by 2 pixels of omitted line.

Offset line

This example alters the *Simple line* to add a perpendicular offset line on the left side of the line, at five pixels distance.

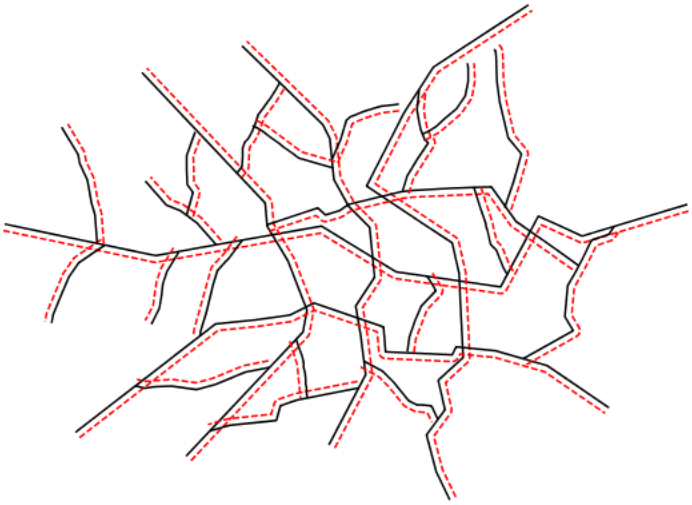


Fig. 6.274: Dashed line

Code

Download the "Offset line" MBStlye

```
{
  "version": 8,
  "name": "simple-offsetline",
  "layers": [
    {
      "id": "simple-line",
      "type": "line",
      "paint": {
        "line-color": "#000000",
        "line-width": 1
      }
    },
    {
      "id": "simple-offsetline",
      "type": "line",
      "paint": {
        "line-color": "#FF0000",
        "line-width": 1,
        "line-dasharray": [5, 2],
        "line-offset": 5
      }
    }
  ]
}
```

Details

In this example, **lines 5-11** draw a simple black line like in the Simple line example. **Lines 13-21** draw a red dashed line like in the above Dashed line example. **Line 20** modifies the dashed line with a 5 pixel offset from the line geometry.

Polygons

Polygons are two dimensional shapes that contain both an outer stroke (or "outline") and an inside (or "fill"). A polygon can be thought of as an irregularly-shaped point and is styled in similar ways to circles.

Example polygons layer

The `polygons layer` used below contains county information for a fictional country. For reference, the attribute table for the polygons is included below.

`fid` (Feature ID)	`name` (County name)	`pop` (Population)
polygon.1	Irony County	412234
polygon.2	Tracker County	235421
polygon.3	Dracula County	135022
polygon.4	Poly County	1567879
polygon.5	Bearing County	201989
polygon.6	Monte Cristo County	152734
polygon.7	Massive County	67123
polygon.8	Rhombus County	198029

`Download the polygons shapefile`

Simple polygon

This example shows a polygon filled in blue.

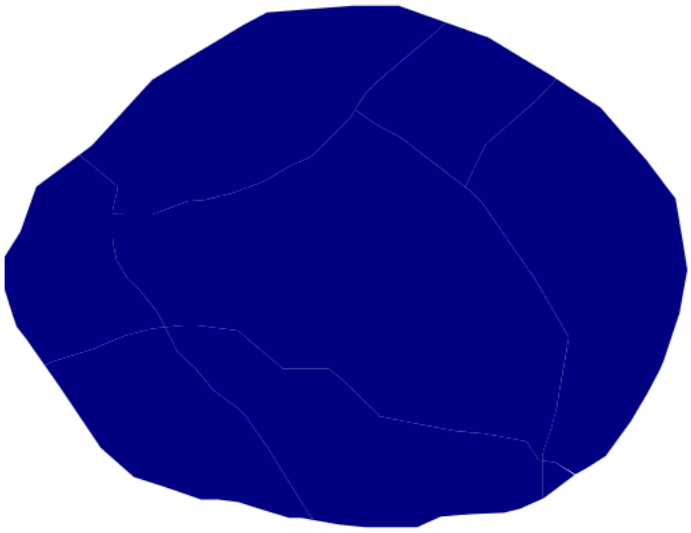


Fig. 6.275: Simple polygon

Code

`Download the "Simple polygon" MBStyle`

```
{
  "version": 8,
  "name": "simple-polygon",
  "layers": [
    {
      "id": "polygon",
      "type": "fill",
      "paint": {
        "fill-color": "#000080"
```

```
10                                    }
11                                 }
12                              ]
13                           }
```

Details

There is one layer for this style, which is the simplest possible situation. Styling polygons is accomplished via the fill type (**line 7**). **Line 9** specifies dark blue (`'#000080'`) as the polygon's fill color.

Note: The light-colored outlines around the polygons in the figure are artifacts of the renderer caused by the polygons being adjacent. There is no outline in this style.

Simple polygon with stroke

This example adds a 1 pixel white outline to the *Simple polygon* example.

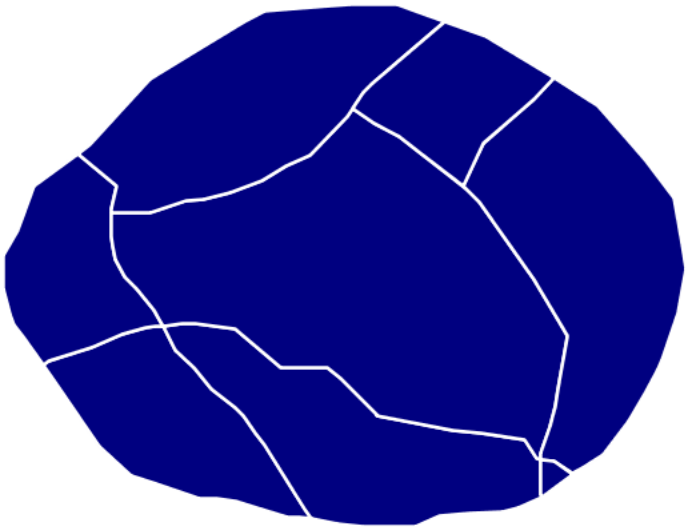


Fig. 6.276: Simple polygon with stroke

Code

Download the "Simple polygon with stroke" MBStyle

```
1                              {
2                                "version": 8,
3                                "name": "simple-polygon-outline",
```

```
4                                   "layers": [
5                                     {
6                                       "id": "polygon-outline",
7                                       "type": "fill",
8                                       "paint": {
9                                         "fill-outline-color": "#FFFFFF",
10                                        "fill-color": "#000080"
11                                      }
12                                    }
13                                  ]
14                                }
```

Details

This example is similar to the *Simple polygon* example above, with the addition of `fill-outline` paint parameter (**line 9**). **Line 9** also sets the color of stroke to white (`'#FFFFFF'`), the `"fill-outline-color"` can only be 1 pixel, a limitation of MBStyle.

Transparent polygon

This example builds on the *Simple polygon with stroke* example and makes the fill partially transparent by setting the opacity to 50%.

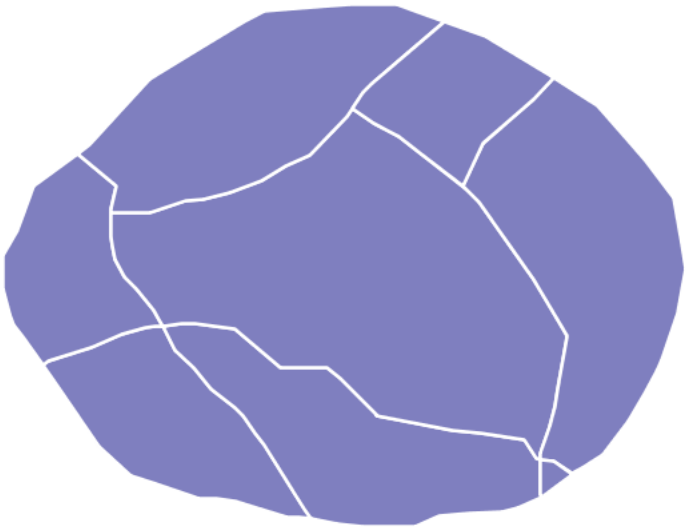


Fig. 6.277: Transparent polygon

Code

`Download the "Transparent polygon" MBStyle`

```
1  {
2    "version": 8,
3    "name": "simple-polygon-transparent",
4    "layers": [
5      {
6        "id": "polygon-transparent",
7        "type": "fill",
8        "paint": {
9          "fill-outline-color": "#FFFFFF",
10         "fill-color": "#000080",
11         "fill-opacity": 0.5
12       }
13     }
14   ]
15 }
```

Details

This example is similar to the *Simple polygon with stroke* example, save for defining the fill's opacity in **line 11**. The value of 0.5 results in partially transparent fill that is 50% opaque. An opacity value of 1 would draw the fill as 100% opaque, while an opacity value of 0 would result in a completely transparent (0% opaque) fill. In this example, since the background is white, the dark blue looks lighter. Were the fill imposed on a dark background, the resulting color would be darker.

6.7 Styling Workshop

This workshop will explore how GeoServer styling can be used for a range of creative effects. This workshop also introduces both the CSS and YSLD extensions, which provide alternate styling languages to SLD.

The following material will be covered in this workshop:

Workshop Setup Workshop materials and setup

Design Overview of map design (i.e. cartography) considerations. Select color palette with colorbrewer.

CSS Styling Workbook Introduction to GeoServer styling followed by easy styling with the CSS module.

YSLD Styling Workbook Introduction to GeoServer styling followed by easy styling with the YSLD module.

MBStyle Styling Workbook Introduction to GeoServer styling followed by easy styling with the MBStyle module.

6.7.1 Workshop Setup

Content:

Extension Install

This workshop course requires GeoServer with a few additional extensions.

- CSS Styling: Quickly and easily generate SLD files
- YSLD Styling: An alternative styling language to SLD
- Importer: Wizard for bulk import of data

On Windows the following is recommended:

- FireFox
- Notepad++

The **CSS extension** is distributed as a supported GeoServer extension. Extensions are unpacked into the `libs` folder of the GeoServer application. The **YSLD extension** is a new addition to geoserver and is distributed as an unsupported GeoServer extension.

Note: In a classroom setting these extensions have already been installed.

Manual Install

To download and install the required extensions by hand:

1. Download geoserver-2.10-M0-css-plugin.zip and geoserver-2.10-M0-css-plugin.zip from:

- Development Release (GeoServer WebSite)

   It is important to download the version that matches the GeoServer you are running.

2. Download the geoserver-2.10-SNAPSHOT-ysld-plugin.zip from:

- Community Builds (GeoServer WebSite)

3. Stop the GeoServer application.
4. Navigate into the `webapps/geoserver/WEB-INF/lib` folder.

   These files make up the running GeoServer application.
5. Unzip the contents of the three zip files into the `lib` folder.
6. Restart the Application Server.
7. Login to the Web Administration application. Select **Styles** from the naviagion menu. Click *Create a new style* and ensure both CSS and YSLD are available in the formats dropdown. Click *Cancel* to return to the **Styles** page without saving.

Course Data

Natural Earth

The Natural Earth dataset is a free collection of vector and raster data published by the North American Cartographic Information Society to encourage mapping.

For this course we will be using the Natural Earth cultural and physical vector layers backed by a raster shaded relief dataset.

Fig. 6.278: Natural Earth

The quickstart Natural Earth styling has been exported from QGIS and cleaned up in uDig for use in GeoServer.

Digital Elevation Model

A digital elevation model records height information for visualisation and analysis. We are using a dataset derived from the USGS GTOPO30 dataset.

Fig. 6.279: Digital Elevation Model

The GeoServer "dem" styling has been used for this dataset.

Configuration

Note: In a classroom setting GeoServer has been preconfigured with the appropriate data directory.

To set up GeoServer yourself:

1. Download and unzip the following into your data directory:

- styling-workshop-vector.zip
- styling-workshop-raster.zip

This will produce a `raster` and `vector` folder referenced in the following steps.

Optional default SLD styles:

- styling-workshop-sld.zip

2. Use the **Importer** to add and publish -

   the following TIF Coverage Stores:

- dem/W100N40.TIF
- ne/ne1/NE1_HR_LC_SR.tif

  the following directories of shape files:

- ne/ne1/physical
- ne/ne1/cultural

Data Type	Workspace	Store Name	Type	Enabled?
	ne	NE1	GeoTIFF	✔
	usgs	DEM	GeoTIFF	✔
	ne	cultural	Directory of spatial files (shapefiles)	✔
	opengeo	cntry_shp	Directory of spatial files (shapefiles)	✔

3. Cleaning up the published vector layers:

- Layer names have been shortened for publication - the `ne_10m_admin_1_states_provinces_lines_ship.shp` is published named `states_provinces_shp`
- Use `EPSG:4326` as the spatial reference system
- Optional: Appropriate SLD styles have been provided (from the uDig project)

Type	Workspace	Store	Layer Name	Enabled?	Native SRS
	ne	cultural	boundary_lines_land	✔	EPSG:4326
	ne	cultural	populated_places	✔	EPSG:4326
	ne	cultural	roads	✔	EPSG:4326
	ne	cultural	states_provinces_lines	✔	EPSG:4326
	ne	cultural	states_provinces_shp	✔	EPSG:4326
	ne	cultural	urban_areas	✔	EPSG:4326

4. To clean up the published raster layers:

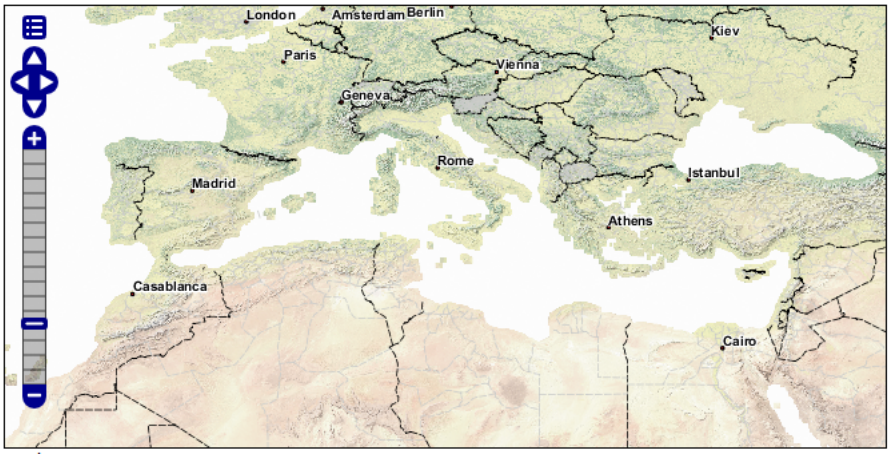
- The NE1 GeoTiff is styled with the default `raster` style
- The usgs:dem GeoTiff is styled with the default `DEM` style

Type	Workspace	Store	Layer Name	Enabled?	Native SRS
	ne	NE1	ne1	✔	EPSG:4326
	usgs	DEM	dem	✔	EPSG:4326

5. Optional: create a `basemap` group layer consisting of:

	Drawing order	Layer	Default Style	Style	Remove
1	⇓	ne:ne1	☐	raster	⊖
2	⇑ ⇓	ne:boundary_lines_land	☐	admin_0_boundary_lines_land	⊖
3	⇑ ⇓	ne:states_provinces_lines	☐	admin_1_states_provinces_lines_shp	⊖
4	⇑	ne:populated_places	☐	populated_places	⊖

   This offers a combined layer, forming a cohesive base map.



6.7.2 Design

This section introduces mapping as a tool for visual communication.

Symbology

In cartography, **symbology is the practice of representing information using shapes, colors and symbols on a map**.

A map legend, offers a quick summary of the symbology used for a map.

The symbology for each layer should be distinct, allowing readers to clearly understand the information being presented. Care should be taken to consider the situation in which the map will be used, such as:

- Screen size of the output device
- Ability of target device to reproduce color
- Allowances for disabilities such as color blindness

Theme

For thematic maps, the symbology is changed on a feature-by-feature basis in order to illustrate the attribute values being presented.

The same data set may be represented in different maps, themed by a different attribute each time.

Using Multiple Themes

A single map can be produced showing two datasets (each themed by a different attribute) allowing readers to look for any interesting patterns.

As an example there may be a relationship between a country's growing region and annual rainfall.

Map Icons

The use of map icons (pictograms, glyphs or symbology sets) is a special case where we have a gap in terminology between Cartography and GIS.

As an example we may wish to represent a point-of-interest data set with each location marked by a different symbol.

In cartography, each "type" is presented to the user as a clearly distinct data set with its own visual representation in the map legend.

This is contrasted with GIS, where the "points of interest" are managed as a single layer and complex styling is used to produce the desired cartographic output.

Technically the data set is themed by an attribute to produce this effect:

- Often an attribute named **type** is introduced, and styling rules are used to associated each value with a distinct graphic mark.
- Another common solution is to distribute the "symbology set" as TrueType font. A character attribute is introduced, the value of which is the appropriate letter to draw.

Map Design

The choice of how to present content is the subject of map design. Cartography, like any venue for design, is a human endeavor between art and science. In this exercise we are going to explore the trade offs in the use of color for effective communication.

It is a challenge to explore cartography without getting stuck in the details of configuring style (which we will cover next). In order to side-step these details, we will be using a web application for this section: **Color Brewer** by Cynthia Brewer of Pennsylvania State University.

Selection of an appropriate color palette is difficult, with a tension between what looks good and what can be understood. The research project that produced the color palettes used by Color Brewer was based on comprehension tests.

1. Navigate to: http://colorbrewer2.org/

   The website provides a generic data set which we can use to determine how effective each choice is in communicating attribute differences. We will be using this website to explore how to effectively theme an attribute.

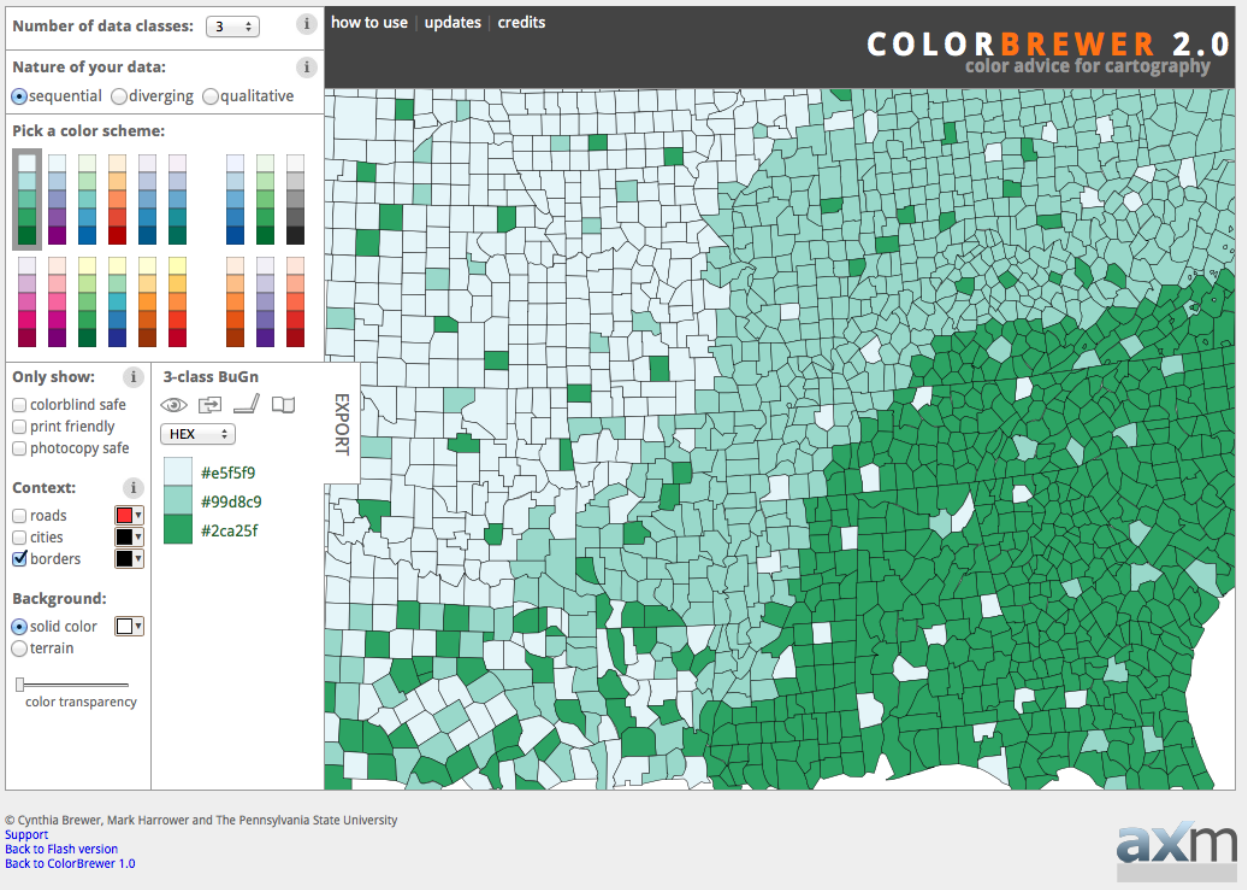


Fig. 6.280: Color Brewer

2. The decisions we make when theming depend entirely on what point we are trying to communicate.

   In this scenario, we are going to communicate a vaccination schedule, county by county. Care should be taken to ensure each county appears equally important, and we should stay clear of red for anyone squeamish about needles. We need to ensure readers can quickly locate their county and look at the appropriate calendar entry.

3. The first step is determining how many attribute values you are looking to communicate. Set *Number of data classes* to `5`.

4. Color brewer offers palettes using three different color schemes:

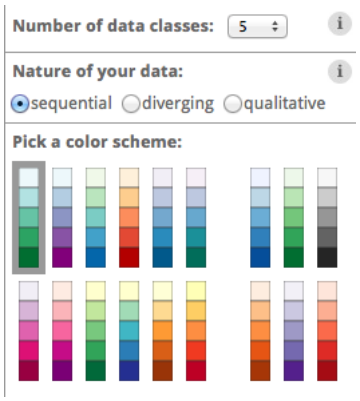
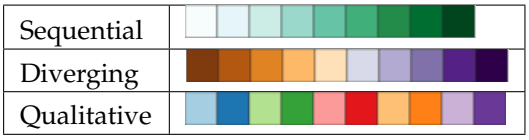


Fig. 6.281: Number of data classes



The nature of our data is qualitative (each attribute value is attached an equal importance, and there is no implied order that wish to communicate with color).

5. Set *Nature of your data* to `Qualitative`. This change drastically reduces the number of color schemes listed.

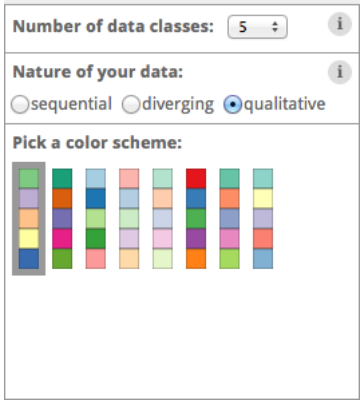


Fig. 6.282: Qualitative color scheme

6. The initial **5-class Accent** color scheme does reasonably well.

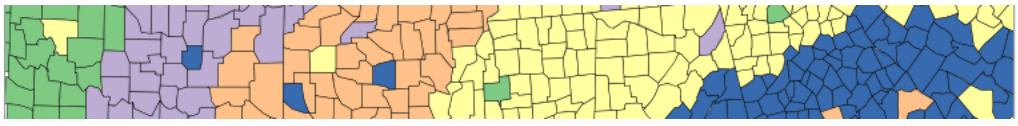


Fig. 6.283: 5-class accent

7. One of our requirements is to help readers locate their county. To assist with that let's turn on roads and cities.

Fig. 6.284: Adding context

8. The map is now starting to look a little busy:

Fig. 6.285: Lots of context

9. Now that we have seen what we are up against, we can try a strategy to help the text and roads stand out while still communicating our vaccination schedule. Change to one of the pastel color schemes.

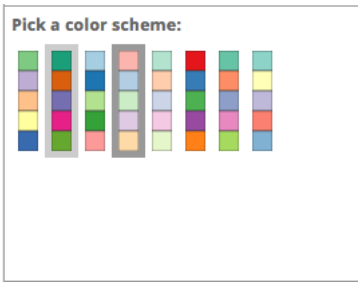


Fig. 6.286: Pastel color scheme

10. Change the borders and roads to gray.
11. The result is fairly clear symbology and provides context.
12. Using our current "pastel" design, set the *Number of data classes* to `9`. At values larger than this, the distinctions between colors becomes so subtle that readers will have trouble clearly distinguishing the content.
13. Make a note of these colors (we will be using them in the exercise on styling next).

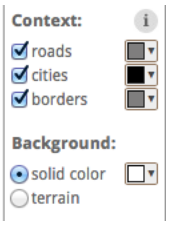


Fig. 6.287: Gray borders and roads

Fig. 6.288: Finished with context

Category	Color
1	#fbb4ae
2	#b3cde3
3	#ccebc5
4	#decbe4
5	#fed9a6
6	#ffffcc
7	#e5d8bd
8	#fddaec
9	#f2f2f2

Bonus

Finished early? While waiting take a moment to explore this topic in more detail, and if you are feeling creative there is a challenge to try.

Note: In a classroom setting please divide the challenges between teams.

This allows us to work through all the material in the time available.

Explore Device Differences

1. Different output devices provide limitations in the amount of color information they can portray.
2. **Explore:** How does changing to a printed map affect the number of classes you can communicate using the current "pastel" approach?

Explore Accessibility

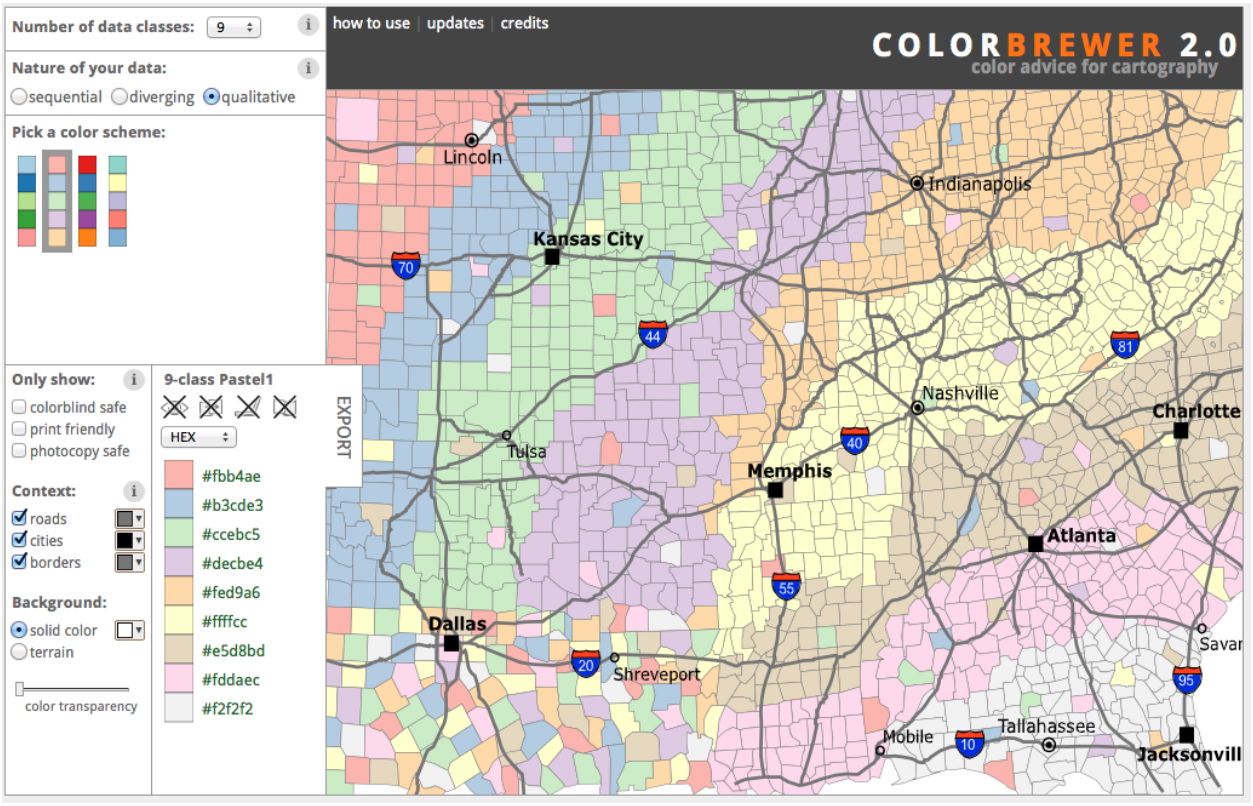


Fig. 6.289: Color palette

1. Communication is a two way street, both in presenting information through design choices, and also perceiving information.

   Disabled readers will have a diminished ability to comprehend maps based on color.

2. **Explore:** What approach can be used to cater to color-blind map readers?

Explore Color Choice

1. The Color Brewer application provides a lot of helpful information using the small "information" icons in each section.

Fig. 6.290: Information icons

2. **Explore:** Using this information which color scheme would you choose for a digital elevation model?

Challenge Adjusted Colour Scheme

1. Some datasets included a critical value or threshold that should be communicated clearly.
2. **Challenge:** How would you adjust a diverging color scheme to be suitable for a digital elevation model that includes bathymetry information (ocean depth)?

   Hint: For a target audience of humans sea-level would be considered a critical value.

Style

The design choices made to represent content is a key aspect of cartography. The style used when rendering data into a visualisation is the result of these choices.

The Open Geospatial Consortium standard for recording style is divided into two parts:

- **Symbology Encoding (SE)**: Records a "feature type style" documenting how individual features are drawn using a series of rules.
- **Style Layer Descriptor (SLD)**: Records which "feature type styles" may be used with a layer.

The **Symbology Encoding** standard provides the terms we will be using to describe style:

- **Stroke**: borders and outlines of shapes
- **Fill**: interior of shapes

Line symbolizer

A line symbolizer documents how individual strokes are used to draw a line string, including color and line width.

The SLD specification provides a default **stroke** used when drawing line strings. These values for color and width will be used if needed.

```
<LineSymbolizer>
  <Stroke/>
</LineSymbolizer>
```

GeoServer includes a default `line.sld` file providing a blue stroke. This file is used when you initially set up a linestring layer.

From GeoServer's **line.sld** style:

```
<LineSymbolizer>
  <Stroke>
    <CssParameter name="stroke">#0000FF</CssParameter>
  </Stroke>
</LineSymbolizer>
```

Polygon symbolizer

A polygon symbolizer documents both the the stroke in addition to the fill used to draw a polygon. A fill can consist of a color, pattern, or other texture:

The SLD specification provides a default gray fill, but does not supply a stroke. These values will be used if you do not provide an alternative.

GeoServer includes a default `polygon.sld` file providing a gray fill and a black outline. This file will be used when you initially create a polygon layer.

From GeoServer's **polygon.sld** style:

```
<PolygonSymbolizer>
  <Fill>
    <CssParameter name="fill">#AAAAAA</CssParameter>
  </Fill>
  <Stroke>
    <CssParameter name="stroke">#000000</CssParameter>
    <CssParameter name="stroke-width">1</CssParameter>
```

```
    </Stroke>
  </PolygonSymbolizer>
```

Point symbolizer

A point symbolizer documents the "mark" used to represent a point. A mark may be defined by a glyph (icon) or a common mark (circle, square, etc.). The point symbolizer records the stroke and fill used to draw the mark.

From GeoServer's default **point.sld** style:

```
<PointSymbolizer>
  <Graphic>
    <Mark>
      <WellKnownName>square</WellKnownName>
      <Fill>
↪
↪       <CssParameter name="fill">#FF0000</CssParameter>
      </Fill>
    </Mark>
    <Size>6</Size>
  </Graphic>
</PointSymbolizer>
```

Text symbolizer

A text symbolizer provides details on how labels are to be drawn, including font, size, and color information.

From the `populated_places.sld` style:

```
<sld:TextSymbolizer>
    <sld:Label>
        <ogc:PropertyName>NAME</ogc:PropertyName>
    </sld:Label>
    <sld:Font>
        <sld:CssParameter␣
↪name="font-family">Arial</sld:CssParameter>
        <sld:CssParameter␣
↪name="font-size">10.0</sld:CssParameter>
        <sld:CssParameter␣
↪name="font-style">normal</sld:CssParameter>
        <sld:CssParameter␣
↪name="font-weight">bold</sld:CssParameter>
    </sld:Font>
    <sld:Halo>
        <sld:Radius>1</sld:Radius>
        <sld:Fill>
            <sld:CssParameter␣
↪name="fill">#FFFFFF</sld:CssParameter>
```

```
            </sld:Fill>
        </sld:Halo>
        <sld:Fill>
            <sld:CssParameter␣
↪name="fill">#000000</sld:CssParameter>
        </sld:Fill>
</sld:TextSymbolizer>
```

Note: The **Style Layer Descriptor** standard makes use of the **Filter Encoding** specification to create small expressions as shown above to access the **NAME** of each city:

```
<ogc:PropertyName>NAME</ogc:PropertyName>
```

This same approach can be used to dynamically generate any values needed for styling.

Raster symbolizer

A raster symbolizer provides a mapping from raster values to colors displayed. This can be provided by a color table, function, or directly mapping bands of data to use for the display channels.

From GeoServer's `dem.sld` style:

```
<RasterSymbolizer>
  <Opacity>1.0</Opacity>
  <ColorMap>
    <ColorMapEntry color="#000000
↪" quantity="-500" label="nodata" opacity="0.0" />
    <ColorMapEntry␣
↪color="#AAFFAA" quantity="0" label="values" />
    <ColorMapEntry color="#00FF00" quantity="1000"/>
    <ColorMapEntry␣
↪color="#FFFF00" quantity="1200" label="values" />
    <ColorMapEntry␣
↪color="#FF7F00" quantity="1400" label="values" />
    <ColorMapEntry␣
↪color="#BF7F3F" quantity="1600" label="values" />
    <ColorMapEntry␣
↪color="#000000" quantity="2000" label="values" />
  </ColorMap>
</RasterSymbolizer>
```

Note: This section uses Open Geospatial Consortium (OGC) terminology in our description of geometry and spatial data. While the terms we use here are general purpose, they may differ slightly from those you are familiar with.

References:

- Geographic Markup Language
- OGC Reference Model (OGC Portal)
- ISO 19107 Geographic information – Spatial schema (ISO)

6.7.3 CSS Styling Workbook

GeoServer styling can be used for a range of creative effects. This section introduces the *CSS Extension* which can be used to quickly generate SLD files.

CSS Quickstart

In the last section, we saw how the OGC defines style using XML documents (called SLD files).

We will now explore GeoServer styling in greater detail using a tool to generate our SLD files. The **Cascading Style Sheet (CSS)** GeoServer extension is used to generate SLD files using a syntax more familiar to web developers.

Using the *CSS extension* to define styles results in shorter examples that are easier to understand. At any point we will be able to review the generated SLD file.

Syntax

This section provides a quick introduction to CSS syntax for mapping professionals who may not be familiar with web design.

Key properties

As we work through CSS styling examples you will note the use of **key properties**. These properties are required to trigger the creation of an appropriate symbolizer in SLD.

stroke	Color (or graphic) for LineString or Polygon border
fill	Color (or graphic) for Polygon Fill
mark	Well-known Mark or graphic used for Point
label	Text expression labeling
halo-radius	Size of halo used to outline label

Using just these key properties and the selector `*`, you will be able to visualize vector data.

For example, here is the key property **stroke** providing a gray representation for line or polygon data:

```
* {
   stroke: gray;
}
```

Here is the key property **fill** providing a blue fill for polygon data:

```
* {
   fill: #2020ED;
}
```

Here is the key property **mark** showing the use of the well-known symbol `square`:

```
* {
   mark: symbol(square);
}
```

Here is the key property **label** generating labels using the `CITY_NAME` feature attribute:

```
* {
   label: [CITY_NAME];
}
```

Here is the key property **halo-radius** providing an outline around generated label:

```
* {
   label: [NAME];
   halo-radius: 1;
}
```

Reference:

- *CSS Cookbook*
- *CSS Examples*

Rules

We have already seen a CSS style composed of a single rule:

```
* {
  mark: symbol(circle);
}
```

We can also make a rule that only applies to a specific FeatureType:

```
populated_places {
  mark: symbol(triangle);
}
```

We can make a style consisting of more than one rule, carefully choosing the selector for each rule. In this case we are using a selector to style capital cities with a star, and non-capital with a circle:

```
[ FEATURECLA = 'Admin-0 capital' ] {
  mark: symbol(star);
  mark-size: 6px;
}

[ FEATURECLA <> 'Admin-0 capital' ] {
  mark: symbol(circle);
  mark-size: 6px;
}
```

The feature attribute test performed above uses **Constraint Query Language (CQL)**. This syntax can be used to define filters to select content, similar to how the SQL WHERE statement is used. It can also be used to define expressions to access attribute values allowing their use when defining style properties.

Rule selectors can also be triggered based on the state of the rendering engine. In this example we are only applying labels when zoomed in:

```
[@scale < 20000000] {
   label: [ NAME ];
}
```

In the above example the label is defined using the CQL Expression `NAME`. This results in a dynamic style that generates each label on a case-by-case basis, filling in the label with the feature attribute `NAME`.

Reference:

- *Filter Syntax*
- *ECQL Reference*

Cascading

In the above example feature attribute selection we repeated information. An alternate approach is to make use of CSS **Cascading** and factor out common properties into a general rule:

```
[ FEATURECLA = 'Admin-0 capital' ] {
  mark: symbol(star);
}

[ FEATURECLA <> 'Admin-0 capital' ] {
  mark: symbol(circle);
}

* {
  mark-size: 6px;
}
```

Pseudo-selector

Up to this point we have been styling individual features, documenting how each shape is represented.

When a shape is represented using a symbol, we have a second challenge: documenting the colors and appearance of the symbol. The CSS extension provides a **pseudo-selector** allowing further properties to be applied to a symbol.

Example of using a pseudo-selector:

```
* {
  mark: symbol(circle);
```

```
}

:mark {
  fill: black;
  stroke: white;
}
```

In this example the `:mark` pseudo-selector is used select the circle mark, and provides a fill and stroke for use when rendering.

Pseudo-selector	Use of symbol
:mark	point markers
:stroke	stroke patterns
:fill	fill patterns
:shield	label shield
:symbol	any use

The above pseudo-selectors apply to all symbols, but to be specific the syntax `nth-symbol(1)` can be used:

```
* {
  mark: symbol(circle);
}

:nth-mark(1) {
  fill: black;
  stroke: white;
}
```

Reference:

- *Styled Marks* (User Guide)

Compare CSS to SLD

The CSS extension is built with the same GeoServer rendering engine in mind, providing access to all the functionality of SLD (along with vendor options for fine control of labeling). The two approaches use slightly different terminology: SLD uses terms familiar to mapping professionals, CSS uses ideas familiar to web developers.

SLD Style

SLD makes use of a series of **Rules** to select content for display. Content is selected using filters that support attribute, spatial and temporal queries.

Once selected, **content is transformed into a shape and drawn using symbolizers**. Symbolizers are configured using CSS Properties to document settings such as "fill" and "opacity".

Content can be drawn by more than one rule, allowing for a range of effects.

Here is an example `SLD file` for reference:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<StyledLayerDescriptor version="1.0.0"
 xsi:schemaLocation="http:/
↪/www.opengis.net/sld StyledLayerDescriptor.xsd"
 xmlns="http://www.opengis.net/sld"
 xmlns:ogc="http://www.opengis.net/ogc"
 xmlns:xlink="http://www.w3.org/1999/xlink"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <NamedLayer>
    <Name>airports</Name>
    <UserStyle>
      <Title>Airports</Title>
      <FeatureTypeStyle>
        <Rule>
          <Name>airports</Name>
          <Title>Airports</Title>
            <PointSymbolizer>
              <Graphic>
                <ExternalGraphic>
                  <OnlineResource xlink:type="simple"
                  xlink:href="airport.svg" />
                  <Format>image/svg</Format>
                </ExternalGraphic>
                <ExternalGraphic>
                  <OnlineResource xlink:type="simple"
                  xlink:href="airport.png" />
                  <Format>image/png</Format>
                </ExternalGraphic>
                <Mark>

↪                <WellKnownName>triangle</WellKnownName>
                  <Fill>

↪      <CssParameter name="fill">#000000</CssParameter>
                  </Fill>
                  <Stroke>

↪    <CssParameter name="stroke">#FFFFFF</CssParameter>
                    <CssParameter
↪name="stroke-opacity">0.50</CssParameter>
                  </Stroke>
                </Mark>
              <Size>16</Size>
            </Graphic>
          </PointSymbolizer>
        </Rule>
      </FeatureTypeStyle>
    </UserStyle>
  </NamedLayer>
</StyledLayerDescriptor>
```

CSS Style

CSS also makes use of rules, each rule making use of **selectors** to shortlist content for display. Each selector uses a CQL filter that

suports attribute, spatial and temporal queries. Once selected, CSS Properties are used to describe how content is rendered.

Content is not drawn by more than one rule. When content satisfies the conditions of more than one rule the resulting properties are combined using a process called inheritance. This technique of having a generic rule that is refined for specific cases is where the **Cascading** in Cascading Style Sheet comes from.

Here is an example using CSS:

```
* {
  mark: url(airport.svg);
  mark-mime: "image/svg";
}
```

In this rule the **selector** `*` is used to match **all content**. The rule defines **properties** indicating how this content is to be styled. The property `mark` is used to indicate we want this content drawn as a **Point**. The value `url(airport.svg)` is a URL reference to the image file used to represent each point. The `mark-mime` property indicates the expected format of this image file.

Tour

To confirm everything works, let's reproduce the airports style above.

1. Navigate to the **Styles** page.
2. Each time we edit a style, the contents of the associated SLD file are replaced. Rather then disrupt any of our existing styles we will create a new style. Click *Add a new style* and choose the following:

Name:	`airport0`
Workspace:	(none specified)
Format:	`CSS`

3. Replace the initial YSLD definition with with our airport CSS example and click *Apply*:

```
* {
  mark: url(airport.svg);
  mark-mime: "image/svg";
}
```

4. Click the *Layer Preview* tab to preview the style. We want to preview on the aiports layer, so click the name of the current layer and select `ne:airports` from the list that appears. You can use the mouse buttons to pan and scroll wheel to change scale.
5. Click *Layer Data* for a summary of the selected data.

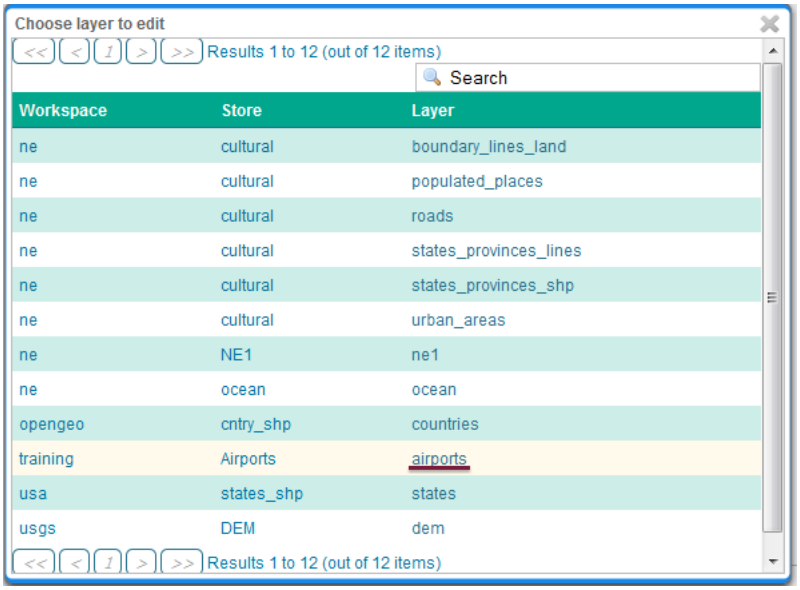


Fig. 6.291: Choosing the airports layer

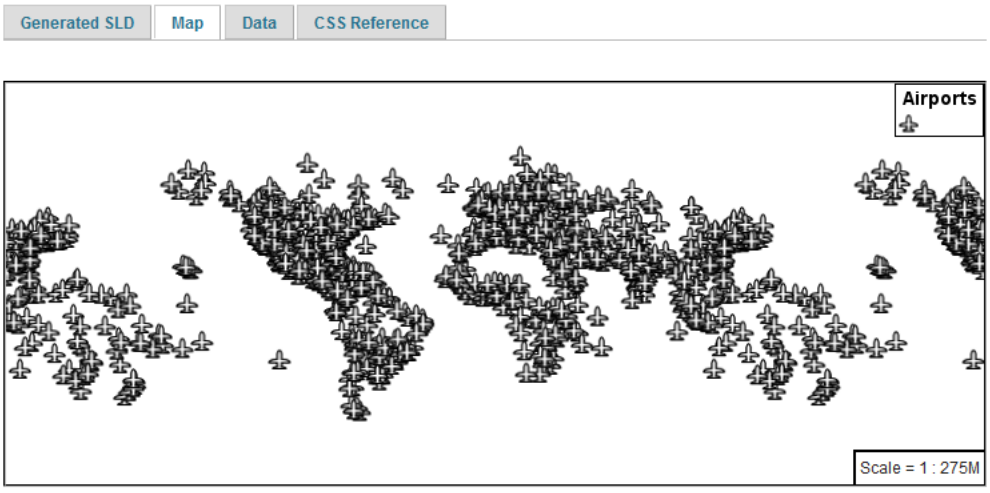
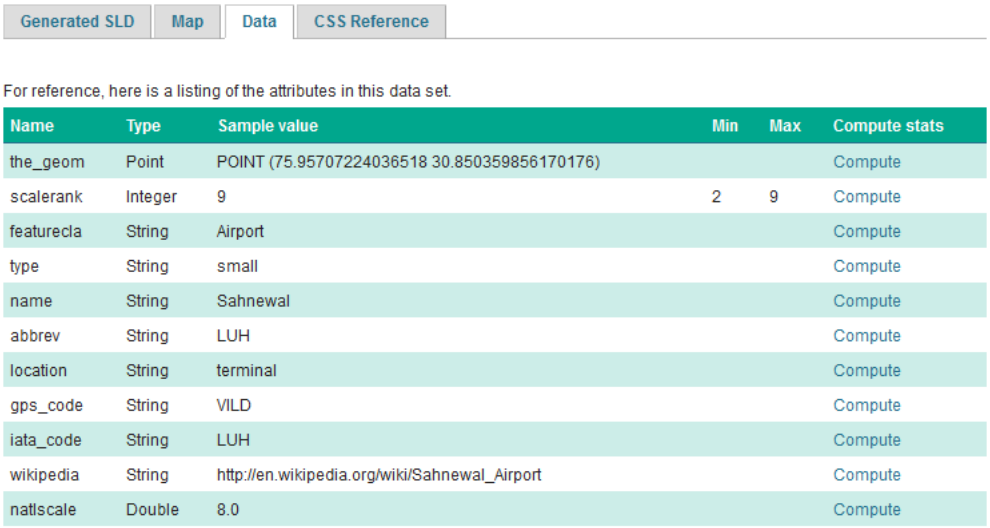


Fig. 6.292: Layer preview



Generated SLD | Map | Data | CSS Reference

For reference, here is a listing of the attributes in this data set.

Name	Type	Sample value	Min	Max	Compute stats
the_geom	Point	POINT (75.95707224036518 30.850359856170176)			Compute
scalerank	Integer	9	2	9	Compute
featurecla	String	Airport			Compute
type	String	small			Compute
name	String	Sahnewal			Compute
abbrev	String	LUH			Compute
location	String	terminal			Compute
gps_code	String	VILD			Compute
iata_code	String	LUH			Compute
wikipedia	String	http://en.wikipedia.org/wiki/Sahnewal_Airport			Compute
natlscale	Double	8.0			Compute

Fig. 6.293: Layer attributes

Bonus

Finished early? For now please help your neighbour so we can proceed with the workshop.

If you are really stuck please consider the following challenge rather than skipping ahead.

Explore Data

1. Return to the *Data* tab and use the *Compute* link to determine the minimum and maximum for the **scalerank** attribute.

Challenge Compare SLD Generation

The rest API can be used to review your CSS file directly.

Browser:

- view-source:http://localhost:8080/geoserver/rest/styles/airport0.css

Command line:

```
curl -v -u admin:geoserver -XGET http:/
↪/localhost:8080/geoserver/rest/styles/airports0.css
```

1. The REST API can also be used generate an SLD file:

Browser:

- view-source:http://localhost:8080/geoserver/rest/styles/airport0.sld?pretty=true

Command line:

```
curl -v -u admin:geoserver -XGET http://localhost:8080/
↪geoserver/rest/styles/airports0.sld?pretty=true
```

1. Compare the generated SLD differ above with the hand written `SLD file` used as an example?

   Challenge: What differences can you spot?

Lines

We will start our tour of CSS styling by looking at the representation of lines.

Fig. 6.294: LineString Geometry

Review of line symbology:

- Lines are used to represent physical details that are too small to be represented at the current scale. Line work can also be used to model non-physical ideas such as network connectivity, or the boundary between land-use classifications. **The visual width of lines do not change depending on scale.**
- Lines are recording as LineStrings or Curves depending on the geometry model used.
- SLD uses a **LineSymbolizer** record how the shape of a line is drawn. The primary characteristic documented is the **Stroke** used to draw each segment between vertices.
- Labeling of line work is anchored to the mid-point of the line. GeoServer provides an option to allow label rotation aligned with line segments.

  For our exercises we are going to be using simple CSS documents, often consisting of a single rule, in order to focus on the properties used for line symbology.

  Each exercise makes use of the `ne:roads` layer.

  Reference:

- *Line Symbology* (User Manual | CSS Property Listing)
- *Lines* (User Manual | CSS Cookbook)
- *LineString* (User Manual | SLD Reference)

Stroke

The only mandatory property for representation of linework is **stroke**. This is a **key property**; its presence triggers the generation of an appropriate LineSymbolizer.

The use of **stroke** as a key property prevents CSS from having the idea of a default line color (as the **stroke** information must be supplied each time).

Fig. 6.295: Basic Stroke Properties

1. Navigate to the **CSS Styles** page.
2. Click *Choose a different layer* and select `ne:roads` from the list.
3. Click *Create a new style* and choose the following:

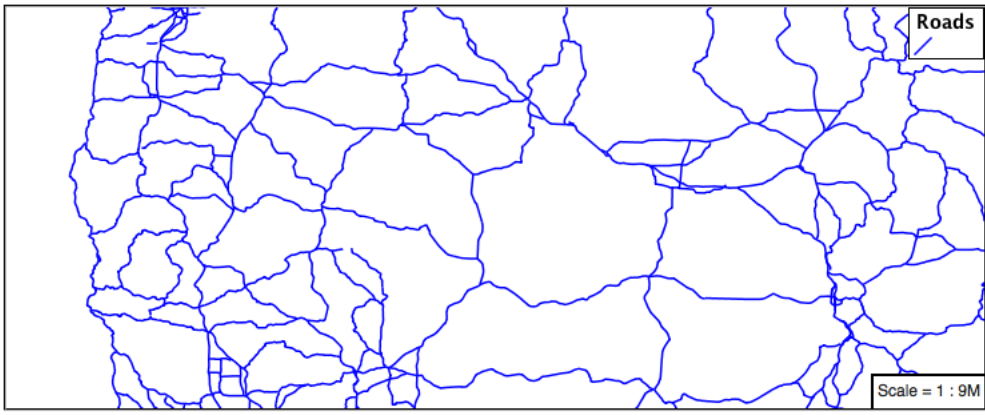
Workspace for new layer:	`No workspace`
New style name:	`line_example`

4. Replace the generated CSS definition with the following **stroke** example:

```
/* @title Line
 * @abstract Example line symbolization
 */
* {
  stroke: blue;
}
```

5. Click *Submit* and then the *Map* tab for an initial preview.

   You can use this tab to follow along as the style is edited, it will refresh each time *Submit* is pressed.



6. You can look at the *SLD* tab at any time to see the generated SLD. Currently it is showing a straight forward LineSymbolizer generated from the CSS **stroke** property:

```
<sld:UserStyle>
   <sld:Name>Default Styler</sld:Name>
   <sld:FeatureTypeStyle>
      <sld:Name>name</sld:Name>
      <sld:Rule>
         <sld:Title>Line</sld:Title>
         <sld:Abstract>
↪Example line symboloization</sld:Abstract>
         <sld:LineSymbolizer>
            <sld:Stroke>
               <sld:CssParameter␣
↪name="stroke">#0000ff</sld:CssParameter>
```

```
                </sld:Stroke>
            </sld:LineSymbolizer>
        </sld:Rule>
    </sld:FeatureTypeStyle>
</sld:UserStyle>
```

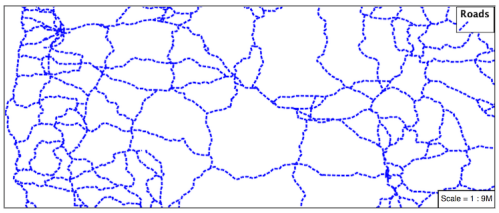
7. Additional properties cane be used fine-tune appearance. Use **stroke-width** to specify the width of the line.

```
/* @title Line
 * @abstract Example line symbolization
 */
 * {
   stroke: blue;
   stroke-width: 2px;
 }
```

8. The **stroke-dasharray** is used to define breaks rendering the line as a dot dash pattern.

```
/* @title Line
 * @abstract Example line symbolization
 */
 * {
   stroke: blue;
   stroke-width: 2px;
   stroke-dasharray: 5 2;
 }
```

9. Check the *Map* tab to preview the result.



Note: The GeoServer rendering engine is quite sophisticated and allows the use of units of measure (such as `m` or `ft`). While we are using pixels in this example, real world units will be converted using the current scale.

Z-Index

The next exercise shows how to work around a limitation when using multiple strokes to render a line.

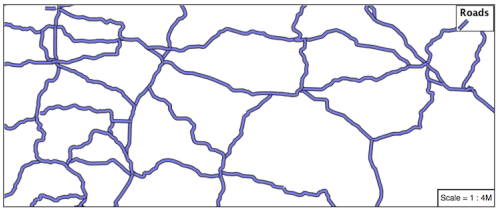
Fig. 6.296: Use of Z-Index

1. Providing two strokes is often used to provide a contrasting edge (called casing) to thick line work.

Update `line_example` with the following:

```
* {
  stroke: black, #8080E6;
  stroke-width: 5px, 3px;
}
```

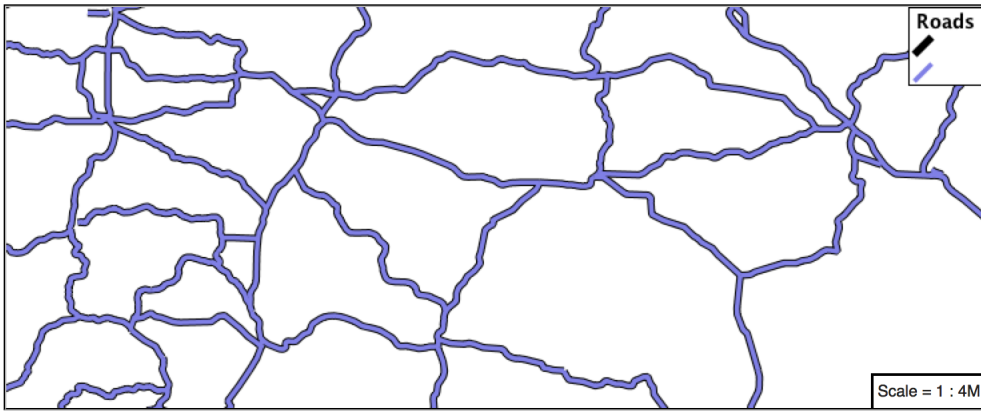
2. If you look carefully you can see a problem with our initial attempt. The junctions of each line show that the casing outlines each line individually, making the lines appear randomly overlapped. Ideally we would like to control this process, only making use of this effect for overpasses.



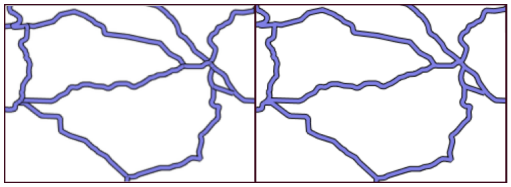
3. The **z-index** parameter allows a draw order to be supplied. This time all the thick black lines are dawn first (at z-index 0) followed by the thinner blue lines (at z-index 1).

```
* {
  stroke: black, #8080E6;
  stroke-width: 5px, 3px;
  z-index: 0, 1;
}
```

4. If you look carefully you can see the difference.



5. By using **z-index** we have been able to simulate line casing.



Label

Our next example is significant as it introduces the how text labels are generated.

Fig. 6.297: Use of Label Property

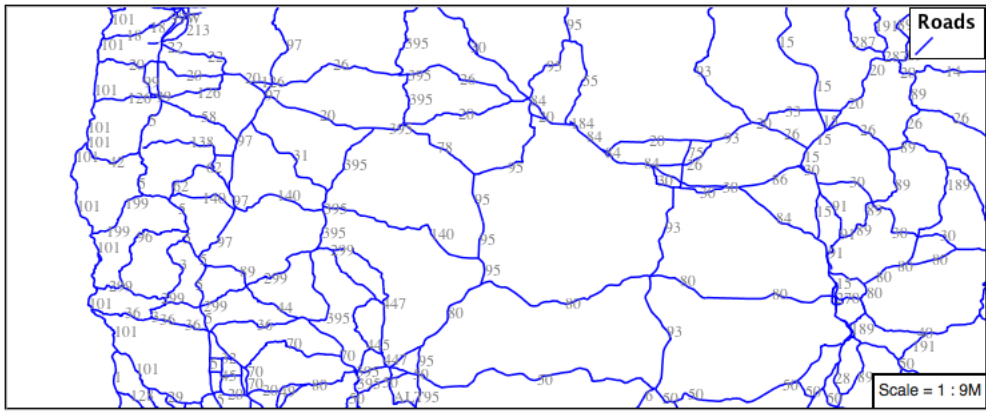
This is also our first example making use of a dynamic style (where the value of a property is defined by an attribute from your data).

1. To enable LineString labeling we will need to use the key properties for both **stroke** and **label**.

   Update `line_example` with the following:

```
* {
  stroke: blue;
  label: [name];
}
```

2. The SLD standard documents the default label position for each kind of Geometry. For LineStrings the initial label is positioned on the midway point of the line.



3. We have used an expression to calculate a property value for label. The **label** property is generated dynamically from the `name` attribute. Expressions are supplied within square brackets, making use of Constraint Query Language (CQL) syntax.

```
* {
  stroke: blue;
  label: [name];
}
```

4. Additional properties can be supplied to fine-tune label presentation:

```
* {
  stroke: blue;
  label: [name];
  font-fill: black;
```

```
    label-offset: 7px;
}
```

5. The **font-fill** property is set to `black` provides the label color.

```
* {
  stroke: blue;
  label: [name];
  font-fill: black;
  label-offset: 7px;
}
```

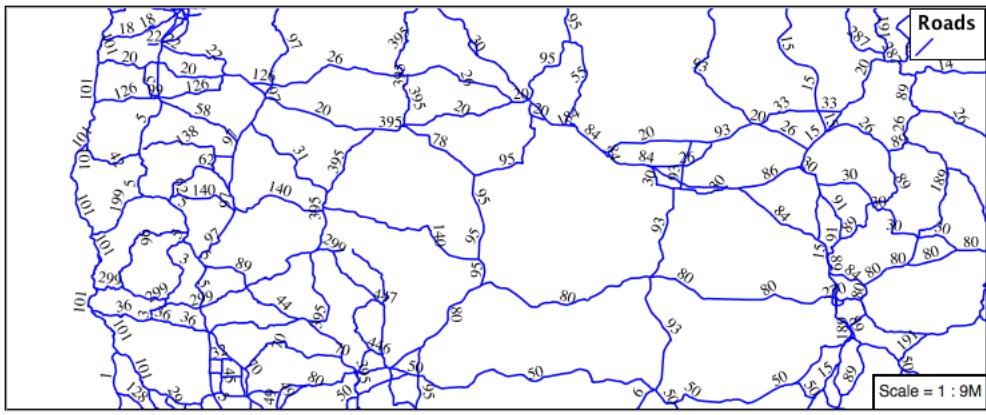
6. The **label-offset** property is used to adjust the starting position used for labeling.

   Normally the displacement offset is supplied using two numbers (allowing an x and y offset from the the midway point used for LineString labeling).

   When labeling a LineString there is a special twist: by specifying a single number for **label-offset** we can ask the rendering engine to position our label a set distance away from the LineString.

```
* {
  stroke: blue;
  label: [name];
  font-fill: black;
  label-offset: 7px;
}
```

7. When used in this manner the rotation of the label will be adjusted automatically to match the LineString.



How Labeling Works

The rendering engine collects all the generated labels during the rendering of each layer. Then, during labeling, the engine sorts through the labels performing collision avoidance (to prevent labels overlapping). Finally the rendering engine draws the labels on top of the map. Even with collision avoidance you can spot areas where labels are so closely spaced that the result is hard to read.

To take greater control over the GeoServer rendering engine we can use extra parameters.

1. The ability to take control of the labeling process is exactly the kind of hint a extra parameter is intended for.

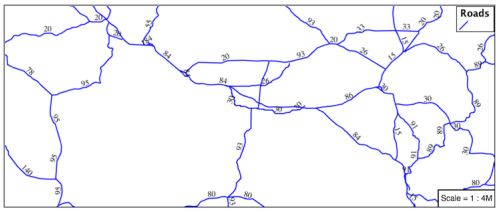
   Update `line_example` with the following:

```
* {
  stroke: blue;
  label: [name];
  font-fill: black;
  label-offset: 7px;
  label-padding: 10;
}
```

2. The parameter **label-padding** provides additional space around our label for use in collision avoidance.

```
* {
  stroke: blue;
  label: [name];
  font-fill: black;
  label-offset: 7px;
  label-padding: 10;
}
```

3. Each label is now separated from its neighbor, improving legibility.



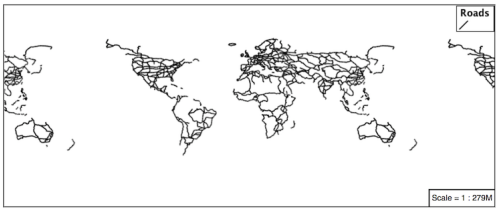
Scale

This section explores the use of attribute selectors and the `@scale` selector together to simplify the road dataset for display.

1. Replace the *line_example* CSS definition with:

```
[scalerank < 4] {
  stroke: black;
}
```

2. And use the *Map* tab to preview the result.



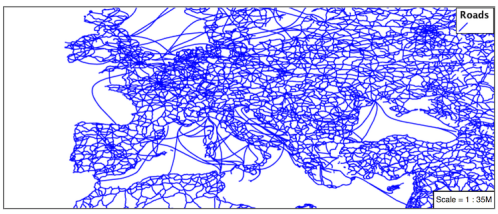
3. The **scalerank** attribute is provided by the Natural Earth dataset to allow control of the level of detail based on scale. Our selector short-listed all content with scalerank 4 or lower, providing a nice quick preview when we are zoomed out.

4. In addition to testing feature attributes, selectors can also be used to check the state of the rendering engine.

   Replace your CSS with the following:

```
[@scale > 35000000] {
   stroke: black;
}
[@scale < 35000000] {
   stroke: blue;
}
```

5. As you adjust the scale in the *Map* preview (using the mouse scroll wheel) the color will change between black and blue. You can read the current scale in the bottom right corner, and the legend will change to reflect the current style.



6. Putting these two ideas together allows control of level detail based on scale:

```
[@scale < 9000000] [scalerank > 7] {
  stroke: #888888;
}

[@scale < 17000000] [scalerank = 7] {
  stroke: #777777;
}

[@scale < 35000000] [scalerank = 6] {
  stroke: #444444;
}

[@scale␣
↪> 9000000] [@scale < 70000000] [scalerank = 5] {
  stroke: #000055;
}
[@scale < 9000000] [scalerank = 5] {
  stroke: #000055;
  stroke-width: 2
}

[@scale > 35000000] [scalerank < 4] {
  stroke: black;
}
[@scale␣
↪> 9000000] [@scale <= 35000000] [scalerank < 4] {
```

```
  stroke: black;
  stroke-width: 2
}
[@scale <= 9000000] [scalerank < 4] {
  stroke: black;
  stroke-width: 4
}
```

7. As shown above selectors can be combined in the same rule:

- Selectors separated by whitespace are combined CQL Filter AND
- Selectors separated by a comma are combined using CQL Filter OR

Our first rule *[@scale < 9000000] [scalerank > 7]* checks that the scale is less than 9M AND scalerank is greater than 7.

Bonus

Finished early? Here are some opportunities to explore what we have learned, and extra challenges requiring creativity and research.

In a classroom setting please divide the challenges between teams (this allows us to work through all the material in the time available).

Explore Follow Line Option

Options can be used to enable some quite useful effects, while still providing a style that can be used by other applications.

1. Update *line_example* with the following:

```
* {
  stroke: #ededff;
  stroke-width: 10;
  label: [level] " " [name];
  font-fill: black;
  label-follow-line: true;
}
```

2. The property **stroke-width** has been used to make our line thicker in order to provide a backdrop for our label.

```
* {
  stroke: #ededff;
  stroke-width: 10;
  label: [level] " " [name];
```

```
    font-fill: black;
    label-follow-line: true;
}
```

3. The **label** property combines combine several CQL expressions together for a longer label.

```
* {
  stroke: #ededff;
  stroke-width: 10;
  label: [level] " " [name];
  font-fill: black;
  label-follow-line: true;
}
```

The combined **label** property:

```
[level] " " [name]
```

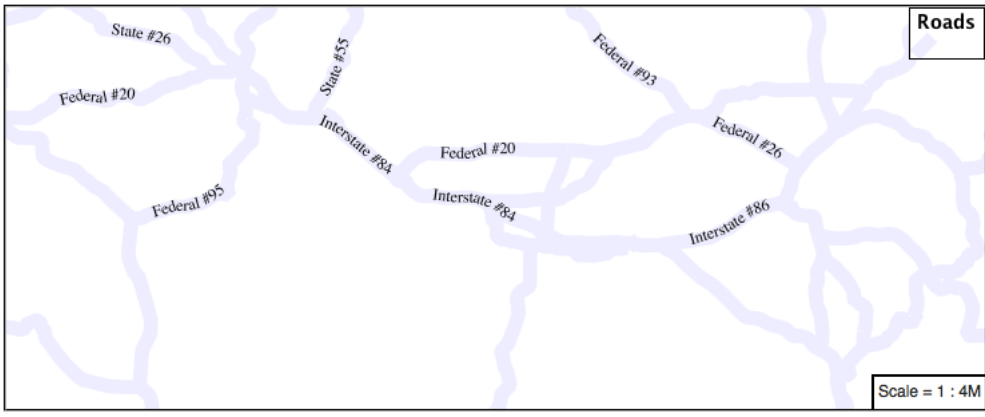
Is internally represented with the **Concatenate** function:

```
[Concatenate(level,' #', name)]
```

4. The property **label-follow-line** provides the ability of have a label exactly follow a LineString character by character.

```
* {
  stroke: #ededff;
  stroke-width: 10;
  label: [level] " " [name];
  font-fill: black;
  label-follow-line: true;
}
```

5. The result is a new appearance for our roads.



Challenge SLD Generation

1. Generate the SLD for the following CSS.

```
* {
  stroke: black;
}
```

What is unusual about the SLD code for this example?

2. Challenge: What is unusual about the generated SLD? Can you explain why it still works as expected?

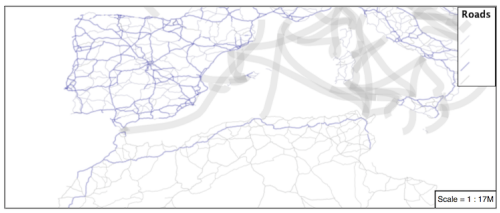
Note: Answer *provided* at the end of the workbook.

Challenge Classification

1. The roads **type** attribute provides classification information.

   You can **Layer Preview** to inspect features to determine available values for type.

2. **Challenge:** Create a new style adjust road appearance based on **type**.



Hint: The available values are 'Major Highway','Secondary Highway','Road' and 'Unknown'.

Note: Answer *provided* at the end of the workbook.

Challenge SLD Z-Index Generation

1. Review the SLD generated by the **z-index** example.

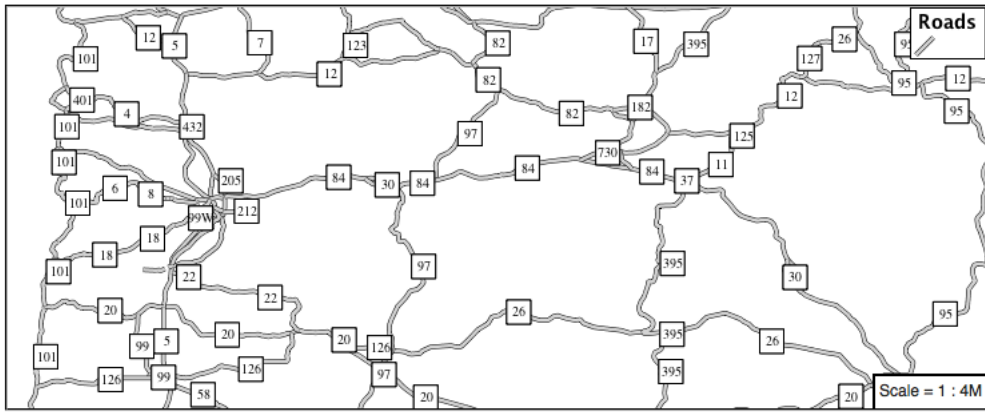
```
* {
  stroke: black, #8080E6;
  stroke-width: 5px, 3px;
  z-index: 0, 1;
}
```

2. *Challenge:* There is an interesting trick in the generated SLD, can you explain how it works?

Note: Answer *provided* at the end of the workbook.

Challenge Label Shields

1. The traditional presentation of roads in the US is the use of a shield symbol, with the road number marked on top.



2. *Challenge:* Have a look at the documentation and reproduce this technique.

Note: Answer *provided* at the end of the workbook.

Polygons

Next we look at how CSS styling can be used to represent polygons.

Fig. 6.298: Polygon Geometry

Review of polygon symbology:

- Polygons offer a direct representation of physical extent or the output of analysis.
- The visual appearance of polygons reflects the current scale.
- Polygons are recorded as a LinearRing describing the polygon boundary. Further LinearRings can be used to describe any holes in the polygon if present.

  The Simple Feature for SQL Geometry model (used by GeoJSON) represents these areas as Polygons, the ISO 19107 geometry model (used by GML3) represents these areas as Surfaces.
- SLD uses a **PolygonSymbolizer** to describe how the shape of a polygon is drawn. The primary characteristic documented is the **Fill** used to shade the polygon interior. The use of a **Stroke** to describe the polygon boundary is optional.
- Labeling of a polygon is anchored to the centroid of the polygon. GeoServer provides a vendor option to allow labels to line wrap to remain within the polygon boundaries.

For our Polygon exercises we will try and limit our CSS documents to a single rule, in order to showcase the properties used for rendering.

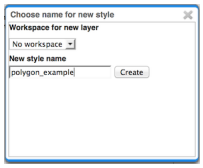
Reference:

- *Polygon Symbology* (User Manual | CSS Property Listing)
- *Polygons* (User Manual | CSS Cookbook)
- *Polygons* (User Manual | SLD Reference)

This exercise makes use of the `ne:states_provinces_shp` layer.

1. Navigate to *Styles*.
2. Create a new style `polygon_example`.

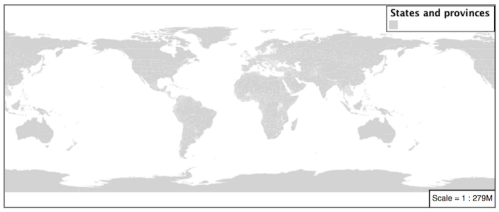
Name:	`polygon_example`
Workspace:	`No workspace`
Format:	`CSS`



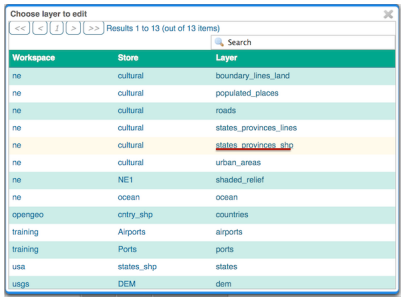
3. Enter the following style and click :menuselection:`Apply` to save:

```
* { fill: lightgrey; }
```

4. Click on the tab *Layer Preview* to preview.



5. Set `ne:states_provinces_shp` as the preview layer.



Stroke and Fill

The **key property** for polygon data is **fill**.

The **fill** property is used to provide the color, or pattern, used to draw the interior of a polygon.

1. Replace the contents of `polygon_example` with the following **fill** example:

```
* {
   fill: gray;
}
```

2. The *Map* tab can be used preview the change:

3. To draw the boundary of the polygon the **stroke** property is used:

   The **stroke** property is used to provide the color, or pattern, for the polygon boundary. It is effected by the same parameters (and vendor specific parameters) as used for LineStrings.

```
* {
   fill: gray;
   stroke: black;
   stroke-width: 2;
}
```

Note: Technically the boundary of a polygon is a specific case of a LineString where the first and last vertex are the same, forming a closed LinearRing.

4. The effect of adding **stroke** is shown in the map preview:

5. An interesting technique when styling polygons in conjunction with background information is to control the fill opacity.

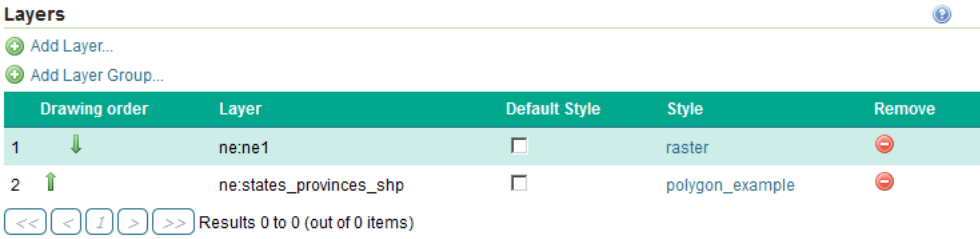
   The **fill-opacity** property is used to adjust transparency (provided as range from 0.0 to 1.0). Use of **fill-opacity** to render polygons works well in conjunction with a raster base map. This approach allows details of the base map to shown through.

   The **stroke-opacity** property is used in a similar fashion, as a range from 0.0 to 1.0.

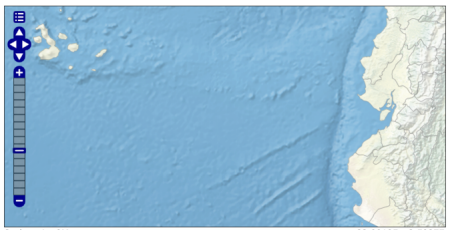
```
* {
   fill: white;
   fill-opacity: 50%;
   stroke: lightgrey;
   stroke-width: 0.25;
   stroke-opacity: 50%;
}
```

6. As shown in the map preview:

7. This effect can be better appreciated using a layer group.



Where the transparent polygons is used lighten the landscape provided by the base map.



Pattern

In addition to color, the **fill** property can also be used to provide a pattern.

The fill pattern is defined by repeating one of the built-in symbols, or making use of an external image.

1. We have two options for configuring a **fill** with a repeating graphic:

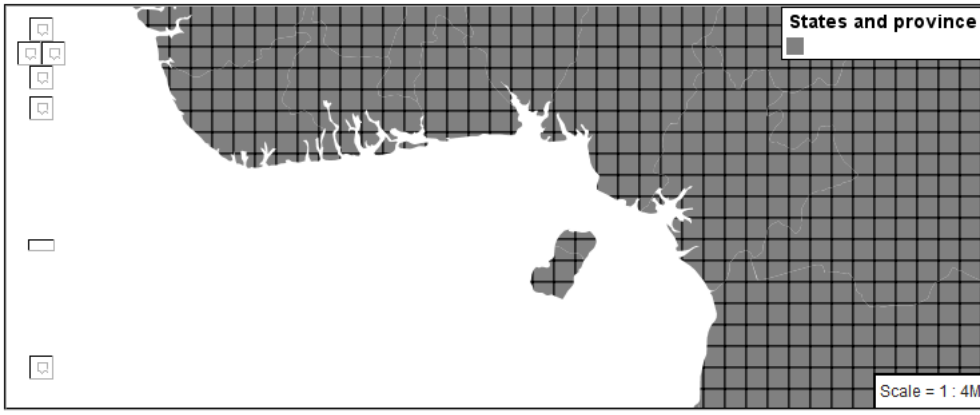
   Using **url** to reference to an external graphic. Used in conjunction with **fill-mime** property.

   Use of **symbol** to access a predefined shape. SLD provides several well-known shapes (circle, square, triangle, arrow, cross, star, and x). GeoServer provides additional shapes specifically for use as fill patterns.

   Update *polygon_example* with the following built-in symbol as a repeating fill pattern:

```
* {
   fill: symbol(square);
}
```

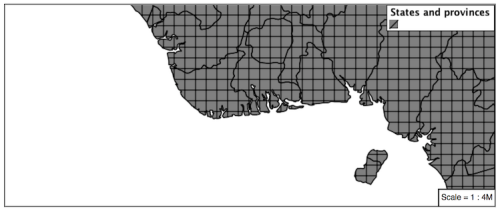
2. The map preview (and legend) will show the result:



3. Add a black stroke:

```
* {
   fill: symbol(square);
   stroke: black;
}
```

4. To outline the individual shapes:



5. Additional fill properties allow control over the orientation and size of the symbol.

   The **fill-size** property is used to adjust the size of the symbol prior to use.

   The **fill-rotation** property is used to adjust the orientation of the symbol.

   Adjust the size and rotation as shown:

```
* {
   fill: symbol(square);
   fill-size: 22px;
   fill-rotation: 45;
   stroke: black;
}
```

6. The size of each symbol is increased, and each symbol rotated by 45 degrees.

Note: Does the above look correct? There is an open request GEOT-4642 to rotate the entire pattern, rather than each individual symbol.

7. The size and rotation properties just affect the size and placement of the symbol, but do not alter the symbol's design. In order to control the color we need to make use of a **pseudo-selector**. We have two options for referencing to our symbol above:

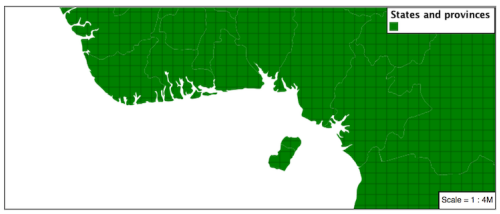
   :symbol provides styling for all the symbols in the CSS document.

   :fill provides styling for all the fill symbols in the CSS document.

8. Replace the contents of `polygon_example` with the following:

```
* {
   fill: symbol(square);
}
:fill {
   fill: green;
   stroke: darkgreen;
}
```

9. This change adjusts the appearance of our grid of squares.



10. If you have more than one symbol:

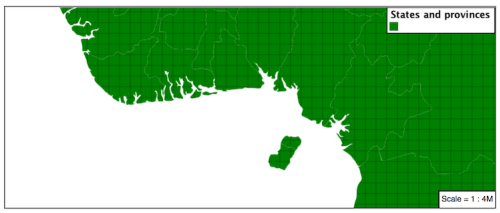
    :nth-symbol(1) is used to specify which symbol in the document we wish to modify.

    :nth-fill(1) provides styling for the indicated fill symbol

    To rewrite our example to use this approach:

```
* {
   fill: symbol(square);
}
:nth-fill(1) {
   fill: green;
   stroke: darkgreen;
}
```

11. Since we only have one fill in our CSS document the map preview looks identical.



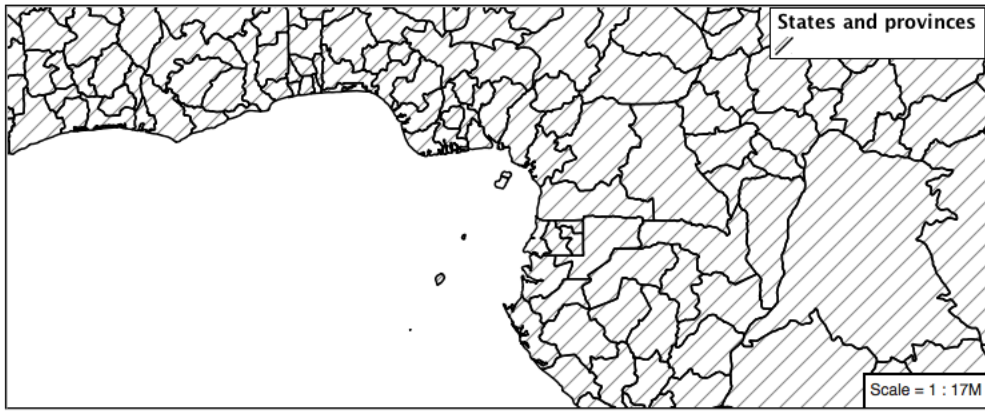
12. The well-known symbols are more suited for marking individual points. Now that we understand how a pattern can be controlled it is time to look at the patterns GeoServer provides.

shape://horizline	horizontal hatching
shape://vertline	vertical hatching
shape://backslash	right hatching pattern
shape://slash	left hatching pattern
shape://plus	vertical and horizontal hatching pattern
shape://times	cross hatch pattern

    Update the example to use **shape://slash** for a pattern of left hatching.

```
* {
   fill: symbol('shape://slash');
   stroke: black;
}
:fill {
  stroke: gray;
}
```

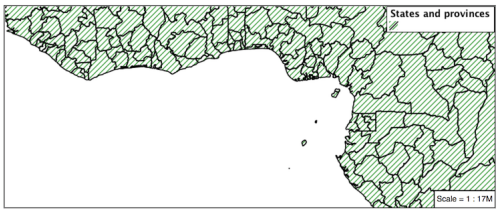
13. This approach is well suited to printed output or low color devices.



14. To control the size of the symbol produced use the **fill-size** property.

```
* {
   fill: symbol('shape://slash');
   fill-size: 8;
   stroke: black;
}
:fill {
   stroke: green;
}
```

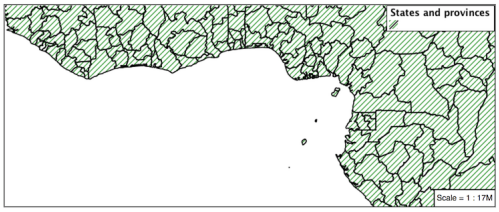
15. This results in a tighter pattern shown:



16. Another approach (producing the same result is to use the **size** property on the appropriate pseudo-selector.

```
* {
   fill: symbol('shape://slash');
   stroke: black;
}
:fill {
   stroke: green;
   size: 8;
}
```

17. This produces the same visual result:



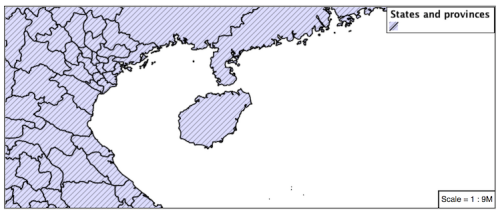
18. Multiple fills can be combined by supplying more than one fill as

part of the same rule.

Note the use of a comma to separate fill-size values (including the first fill-size value which is empty). This was the same approach used when combining strokes.

```
* {
   fill: #DDDDFF, symbol('shape://slash');
   fill-size: '','8';
   stroke: black;
}
:fill {
   stroke: black;
   stroke-width: 0.5;
}
```

19. The resulting image has a solid fill, with a pattern drawn overtop.



Label

Labeling polygons follows the same approach used for LineStrings.

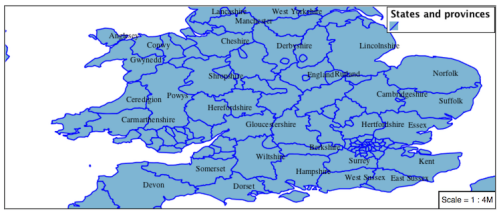
The key properties **fill** and **label** are used to enable Polygon label generation.

1. By default labels are drawn starting at the centroid of each polygon.

2. Try out **label** and **fill** together by replacing our `polygon_example` with the following:

```
* {
  stroke: blue;
  fill: #7EB5D3;
  label: [name];
  font-fill: black;
}
```

3. Each label is drawn from the lower-left corner as shown in the `Map` preview.



4. We can adjust how the label is drawn at the polygon centroid.

   The property **label-anchor** provides two numbers expressing how a label is aligned with respect to the centroid. The first value controls the horizontal alignment, while the second value controls the vertical alignment. Alignment is expressed between 0.0 and 1.0 as shown in the following table.

	Left	Center	Right
Top	0.0 1.0	0.5 1.0	1.0 1.0
Middle	0.0 0.5	0.5 0.5	1.0 0.5
Bottom	0.0 0.0	0.5 0.0	1.0 0.0

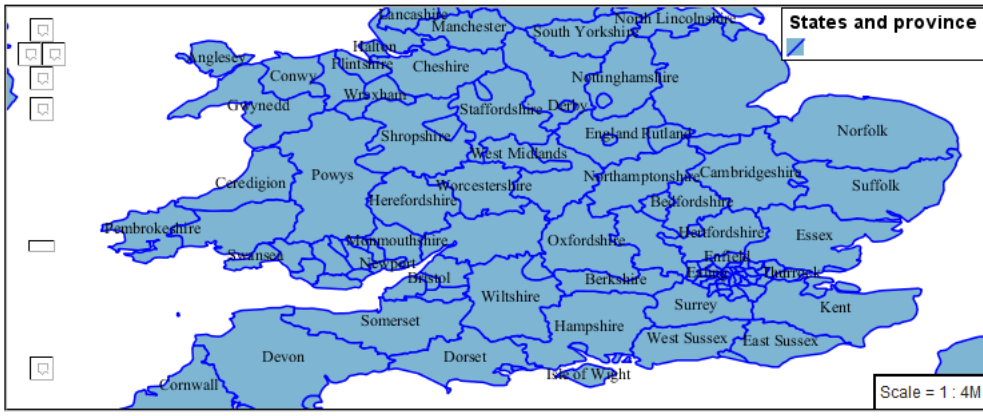
   Adjusting the **label-anchor** is the recommended approach to positioning your labels.

5. Using the **label-anchor** property we can center our labels with respect to geometry centroid.

   To align the center of our label we select 50% horizontally and 50% vertically, by filling in 0.5 and 0.5 below:

```
* {  stroke: blue;
     fill: #7EB5D3;
     label: [name];
     font-fill: black;
     label-anchor: 0.5 0.5;
}
```

6. The labeling position remains at the polygon centroid. We adjust alignment by controlling which part of the label we are "snapping" into position.

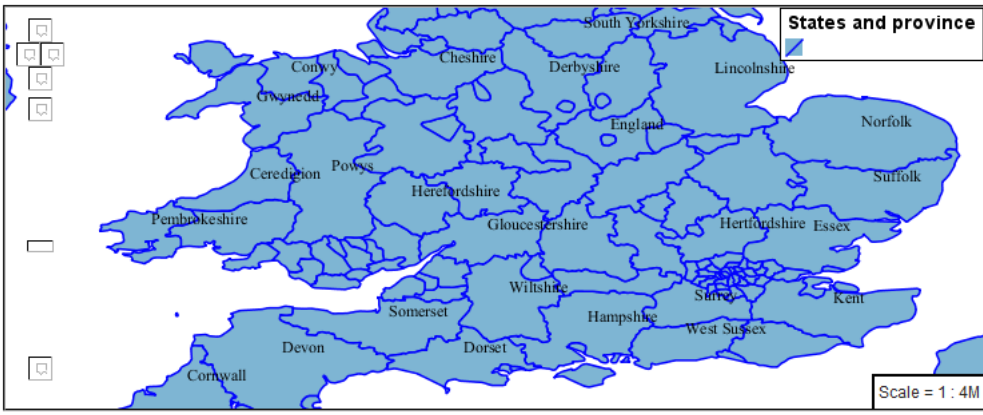


7. The property **label-offset** can be used to provide an initial displacement using and x and y offset.

8. This offset is used to adjust the label position relative to the geometry centroid resulting in the starting label position.

```
* {  stroke: blue;
    fill: #7EB5D3;
    label: [name];
    font-fill: black;
    label-offset: 0 7;
}
```

9. Confirm this result in the map preview.



10. These two settings can be used together.

    The rendering engine starts by determining the label position generated from the geometry centroid and the **label-offset** displacement. The bounding box of the label is used with the **label-anchor** setting align the label to this location.

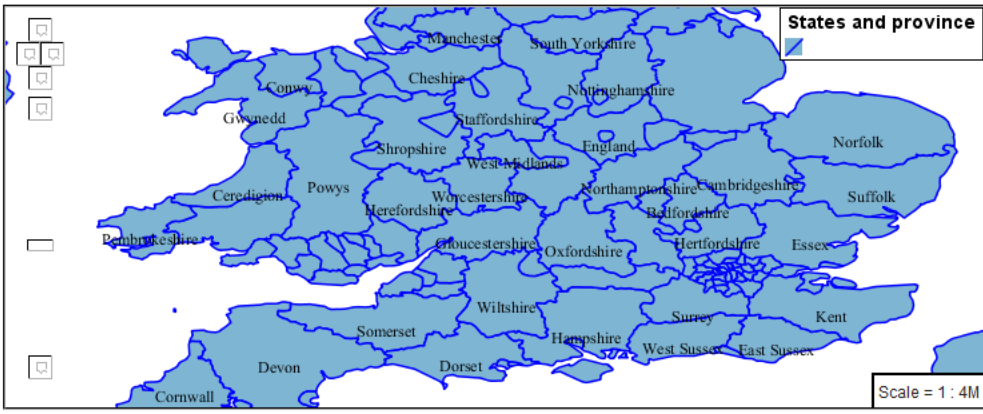
    Step 1: starting label position = centroid + displacement

    Step 2: snap the label anchor to the starting label position

11. To move our labels down (allowing readers to focus on each shape) we can use displacement combined with followed by horizontal alignment.

```
* {  stroke: blue;
    fill: #7EB5D3;
    label: [name];
    font-fill: black;
    label-anchor: 0.5 1;
    label-offset: 0 -7;
 }
```

12. As shown in the map preview.



Legibility

When working with labels a map can become busy very quickly, and difficult to read.

1. GeoServer provides extensive vendor parameters directly controlling the labelling process.

   Many of these parameters focus on controlling conflict resolution (when labels would otherwise overlap).

2. Two common properties for controlling labeling are:

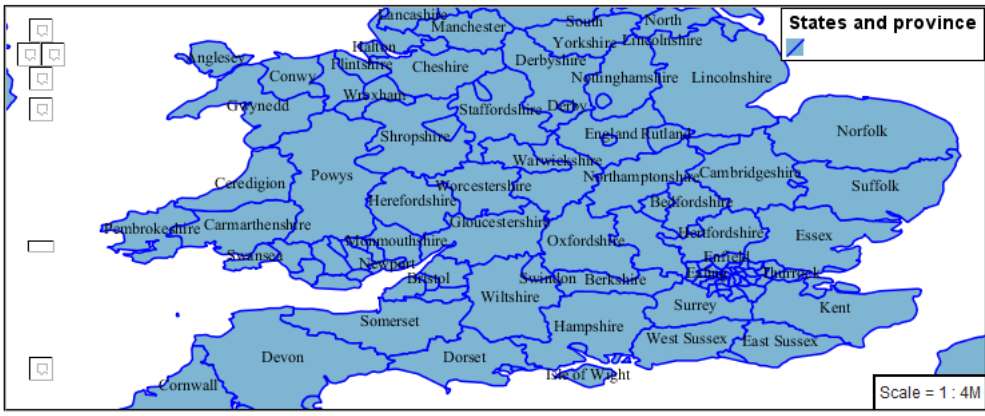
   label-max-displacement indicates the maximum distance GeoServer should displace a label during conflict resolution.

   label-auto-wrap allows any labels extending past the provided width will be wrapped into multiple lines.

3. Using these together we can make a small improvement in our example:

```
* {  stroke: blue;
    fill: #7EB5D3;
    label: [name];
    font-fill: black;
    label-anchor: 0.5 0.5;

    label-max-displacement: 40;
    label-auto-wrap: 70;
  }
```

4. As shown in the following preview.

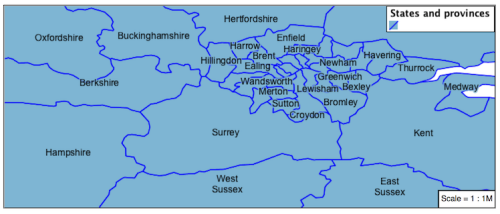


5. Even with this improved spacing between labels, it is difficult to read the result against the complicated line work.

   Use of a halo to outline labels allows the text to stand out from an otherwise busy background. In this case we will make use of the fill color, to provide some space around our labels. We will also change the font to Arial.

```
* {   stroke: blue;
     fill: #7EB5D3;
     label: [name];
     label-anchor: 0.5 0.5;
     font-fill: black;
     font-family: "Arial";
     font-size: 14;
     halo-radius: 2;
     halo-color: #7EB5D3;
     halo-opacity:0.8;

     label-max-displacement: 40;
     label-auto-wrap: 70;
  }
```

6. By making use of **halo-opacity** we we still allow stroke information to show through, but prevent the stroke information from making the text hard to read.



7. And advanced technique for manually taking control of conflict resolution is the use of the **label-priority**.

   This property takes an expression which is used in the event of a conflict. The label with the highest priority "wins."

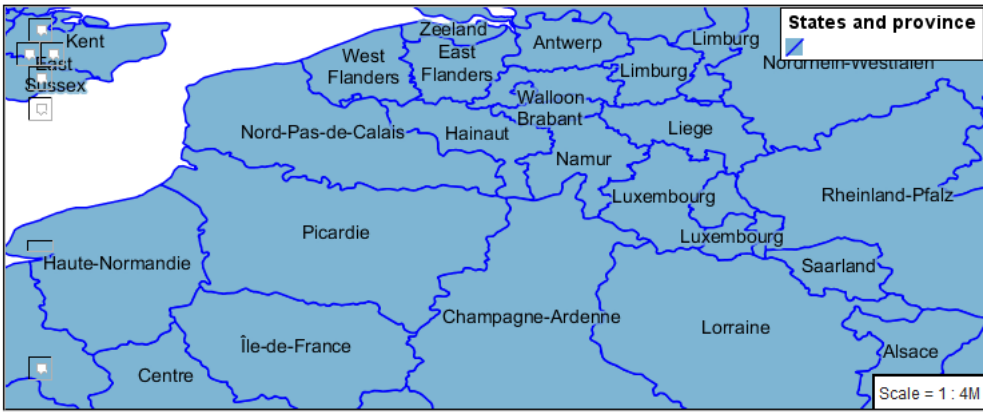
8. The Natural Earth dataset we are using includes a **labelrank** intended to control what labels are displayed based on zoom level.

   The values for **labelrank** go from 0 (for zoomed out) to 20 (for

zoomed in). To use this value for **label-priority** we need to swap the values around so a **scalerank** of 1 is given the highest priority.

```
* {  stroke: blue;
   fill: #7EB5D3;
   label: [name];
   label-anchor: 0.5 0.5;
   font-fill: black;
   font-family: "Arial";
   font-size: 14;
   halo-radius: 2;
   halo-color: #7EB5D3;
   halo-opacity:0.8;

   label-max-displacement: 40;
   label-auto-wrap: 70;
   label-priority: [20-labelrank];
}
```

9. In the following map `East Flanders` will take priority over `Zeeland` when the two labels overlap.



Theme

A thematic map (rather than focusing on representing the shape of the world) uses elements of style to illustrate differences in the data under study. This section is a little more advanced and we will take the time to look at the generated SLD file.

1. We can use a site like ColorBrewer to explore the use of color theming for polygon symbology. In this approach the the fill color of the polygon is determined by the value of the attribute under study.

This presentation of a dataset is known as "theming" by an attribute.

2. For our `ne:states_provinces_shp` dataset, a **mapcolor9** attribute has been provided for this purpose. Theming by **mapcolor9** results in a map where neighbouring countries are visually distinct.

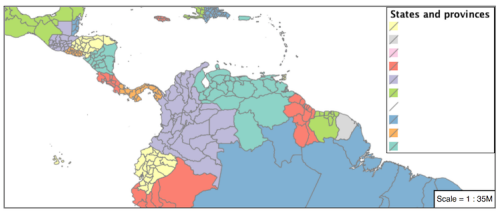
Qualitative 9-class Set3		
#8dd3c7	#fb8072	#b3de69
#ffffb3	#80b1d3	#fccde5
#bebada	#fdb462	#d9d9d9

If you are unfamiliar with theming you may wish to visit http://colorbrewer2.org to learn more. The **i** icons provide an adequate background on theming approaches for qualitative, sequential and diverging datasets.

3. The first approach we will take is to directly select content based on **colormap**, providing a color based on the **9-class Set3** palette above:

```
[mapcolor9=1] {
   fill: #8dd3c7;
}
[mapcolor9=2] {
   fill: #ffffb3;
}
[mapcolor9=3] {
   fill: #bebada;
}
[mapcolor9=4] {
   fill: #fb8072;
}
[mapcolor9=5] {
   fill: #80b1d3;
}
[mapcolor9=6] {
   fill: #fdb462;
}
[mapcolor9=7] {
   fill: #b3de69;
}
[mapcolor9=8] {
   fill: #fccde5;
}
[mapcolor9=9] {
   fill: #d9d9d9;
}
* {
  stroke: gray;
  stroke-width: 0.5;
}
```

4. The *Map* tab can be used to preview this result.



5. This CSS makes use of cascading to avoid repeating the **stroke** and **stroke-width** information multiple times.

As an example the `mapcolor9=2` rule, combined with the `*` rule results in the following collection of properties:

```
[mapcolor9=2] {
  fill: #ffffb3;
  stroke: gray;
  stroke-width: 0.5;
}
```

6. Reviewing the generated SLD shows us this representation:

```
<sld:Rule>
   <ogc:Filter>
      <ogc:PropertyIsEqualTo>
         <ogc:PropertyName>mapcolor9</ogc:PropertyName>
         <ogc:Literal>2</ogc:Literal>
      </ogc:PropertyIsEqualTo>
   </ogc:Filter>
   <sld:PolygonSymbolizer>
      <sld:Fill>
         <sld:CssParameter␣
↪name="fill">#ffffb3</sld:CssParameter>
      </sld:Fill>
   </sld:PolygonSymbolizer>
   <sld:LineSymbolizer>
      <sld:Stroke>
         <sld:CssParameter␣
↪name="stroke">#808080</sld:CssParameter>
         <sld:CssParameter␣
↪name="stroke-width">0.5</sld:CssParameter>
      </sld:Stroke>
   </sld:LineSymbolizer>
</sld:Rule>
```

7. There are three important functions, defined by the Symbology Encoding specification, that are often easier to use for theming than using rules.

- **Recode**: Used the theme qualitative data. Attribute values are directly mapped to styling property such as **fill** or **stroke-width**.
- **Categorize**: Used the theme quantitative data. Categories are defined using min and max ranges, and values are sorted into the appropriate category.
- **Interpolate**: Used to smoothly theme quantitative data by calculating a styling property based on an attribute value.

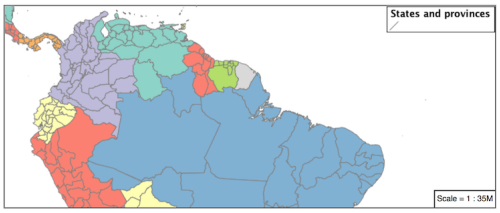
  Theming is an activity, producing a visual result allow map readers to learn more about how an attribute is distributed spatially. We are free to produce this visual in the most efficient way possible.

8. Swap out **mapcolor9** theme to use the **Recode** function:

```
* {
  fill:[
    recode(mapcolor9,
      1,'#8dd3c7', 2,'#ffffb3', 3,'#bebada',
      4,'#fb8072', 5,'#80b1d3', 6,'#fdb462',
      7,'#b3de69', 8,'#fccde5', 9,'#d9d9d9')
```

```
    ];
    stroke: gray;
    stroke-width: 0.5;
}
```

9. The *Map* tab provides the same preview.



10. The *Generated SLD* tab shows where things get interesting. Our generated style now consists of a single **Rule**:

```
<sld:Rule>
   <sld:PolygonSymbolizer>
      <sld:Fill>
         <sld:CssParameter name="fill">
            <ogc:Function name="Recode">
↪          <ogc:PropertyName>mapcolor9</ogc:PropertyName>
               <ogc:Literal>1</ogc:Literal>
                  <ogc:Literal>#8dd3c7</ogc:Literal>
               <ogc:Literal>2</ogc:Literal>
                  <ogc:Literal>#ffffb3</ogc:Literal>
               <ogc:Literal>3</ogc:Literal>
                  <ogc:Literal>#bebada</ogc:Literal>
               <ogc:Literal>4</ogc:Literal>
                  <ogc:Literal>#fb8072</ogc:Literal>
               <ogc:Literal>5</ogc:Literal>
                  <ogc:Literal>#80b1d3</ogc:Literal>
               <ogc:Literal>6</ogc:Literal>
                  <ogc:Literal>#fdb462</ogc:Literal>
               <ogc:Literal>7</ogc:Literal>
                  <ogc:Literal>#b3de69</ogc:Literal>
               <ogc:Literal>8</ogc:Literal>
                  <ogc:Literal>#fccde5</ogc:Literal>
               <ogc:Literal>9</ogc:Literal>
                  <ogc:Literal>#d9d9d9</ogc:Literal>
            </ogc:Function>
            </sld:CssParameter>
      </sld:Fill>
   </sld:PolygonSymbolizer>
   <sld:LineSymbolizer>
      <sld:Stroke>
         <sld:CssParameter
↪name="stroke">#808080</sld:CssParameter>
         <sld:CssParameter
↪name="stroke-width">0.5</sld:CssParameter>
      </sld:Stroke>
   </sld:LineSymbolizer>
</sld:Rule>
```

Bonus

The following optional explore and challenge activities offer a chance to review and apply the ideas introduced here. The challenge activities equire a bit of creativity and research to complete.

In a classroom setting you are encouraged to team up into groups, with each group taking on a different challenge.

Explore Antialiasing

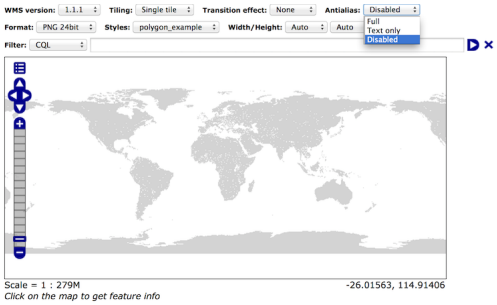
1. When we rendered our initial preview, without a stroke, thin white gaps (or slivers) are visible between our polygons.

   This effect is made more pronounced by the rendering engine making use of the Java 2D sub-pixel accuracy. This technique is primarily used to prevent an aliased (stair-stepped) appearance on diagonal lines.

2. Clients can turn this feature off using a GetMap format option:

```
format_options=antialiasing=off;
```

   The **LayerPreview** provides access to this setting from the Open Layers **Options Toolbar**:



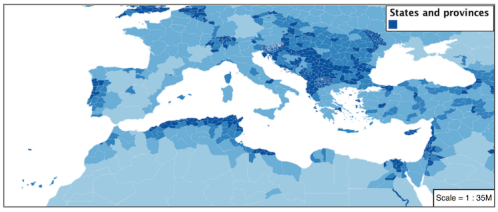
3. **Explore:** Experiment with **fill** and **stroke** settings to eliminate slivers between polygons.

Note: Answer *provided* at the end of the workbook.

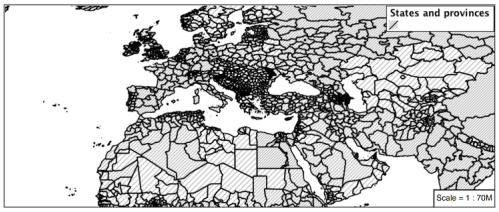
Explore Categorize

1. The **Categorize** function can be used to generate property values based on quantitative information. Here is an example using Categorize to color states according to size.

```
* {
   fill: [
      Categorize(Shape_Area,
         '#08519c', 0.5,
         '#3182bd', 1,
         '#6baed6', 5,
         '#9ecae1', 60,
         '#c6dbef', 80,
         '#eff3ff')
   ];
}
```



2. An exciting use of the GeoServer **shape** symbols is the theming by changing the **fill-size** used for pattern density.
3. **Explore:** Use the **Categorize** function to theme by **datarank**.



Note: Answer *provided* at the end of the workbook.

Challenge Goodness of Fit

1. A subject we touched on during labeling was the conflict resolution GeoServer performs to ensure labels do not overlap.
2. In addition to the vendor parameter for max displacement you can experiment with different values for "goodness of fit". These settings control how far GeoServer is willing to move a label to avoid conflict, and under what terms it simply gives up:

```
label-fit-goodness: 0.3;
label-max-displacement: 130;
```

3. You can also experiment with turning off this facility completely:

```
label-conflict-resolution: false;
```

4. **Challenge:** Construct your own example using max displacement and fit-goodness.

Challenge Halo

1. The halo example used the fill color and opacity for a muted halo, while this improved readability it did not bring attention to our labels.

   A common design choice for emphasis is to outline the text in a contrasting color.

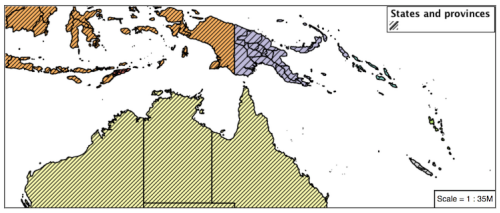
2. **Challenge:** Produce a map that uses a white halo around black text.

Note: Answer *provided* at the end of the workbook.

Challenge Theming using Multiple Attributes

1. A powerful tool is theming using multiple attributes. This is an important concept allowing map readers to perform "integration by eyeball" (detecting correlations between attribute values information).

2. **Challenge:** Combine the **mapcolor9** and **datarank** examples to reproduce the following map.

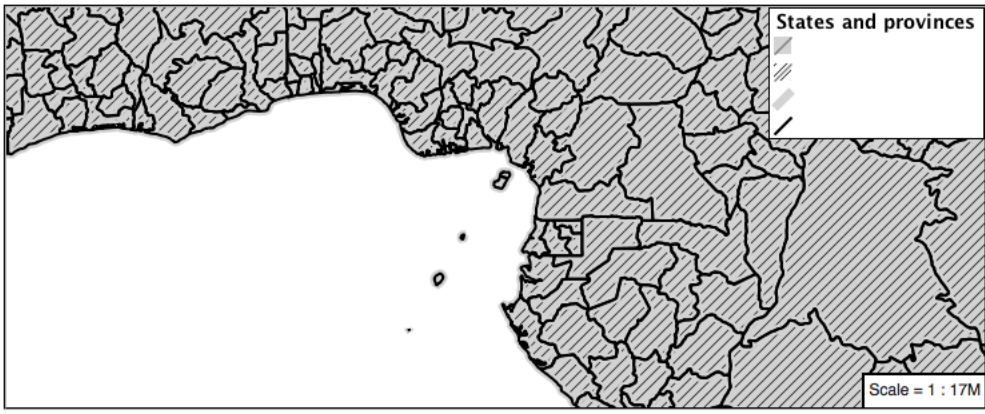


Note: Answer *provided* at the end of the workbook.

Challenge Use of Z-Index

1. Earlier we looked at using **z-index** to simulate line string casing. The line work was drawn twice, once with thick line, and then a second time with a thinner line. The resulting effect is similar to text halos - providing breathing space around complex line work allowing it to stand out.

2. **Challenge:** Use what you know of LineString **z-index** to reproduce the following map:



Note: Answer *provided* at the end of the workbook.

Points

The next stop of the CSS styling tour is the representation of points.

Review of point symbology:

- Points are used to represent a location only, and do not form a shape. The visual width of lines do not change depending on scale.
- SLD uses a **PointSymbolizer** record how the shape of a line is drawn.
- Labeling of points is anchored to the point location.

  As points have no inherent shape of of their own, emphasis is placed on marking locations with an appropriate symbol.

Reference:

- *Point Symbology* (User Manual | CSS Property Listing)
- *Points* (User Manual | CSS Cookbook)
- *Styled Marks* (User Manual | CSS Styling)
- *Point* (User Manual | SLD Reference)

This exercise makes use of the `ne:populated_places` layer.

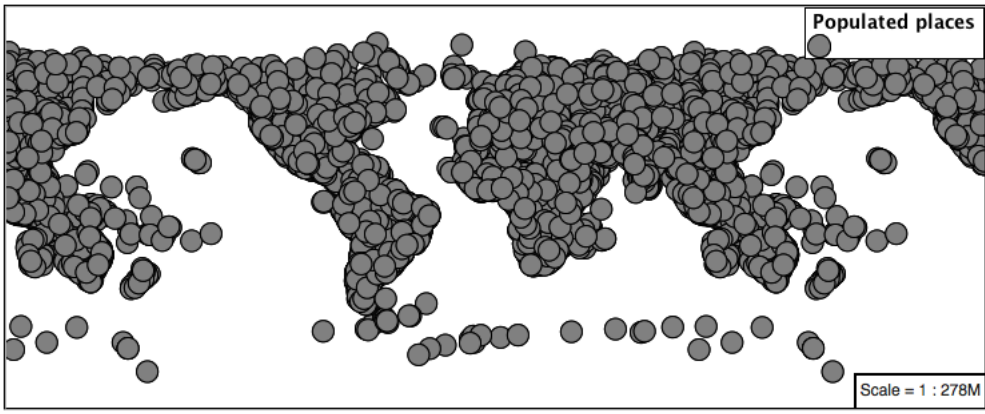
1. Navigate to the **Styles** page.
2. Click *Add a new style* and choose the following:

Name:	`point_example`
Workspace:	`No workspace`
Format:	`CSS`

3. Replace the initial CSS definition with the following and click *apply*:

```
* {
  mark: symbol(circle);
}
```

4. And use the *Layer Preview* tab to preview the result.



Mark

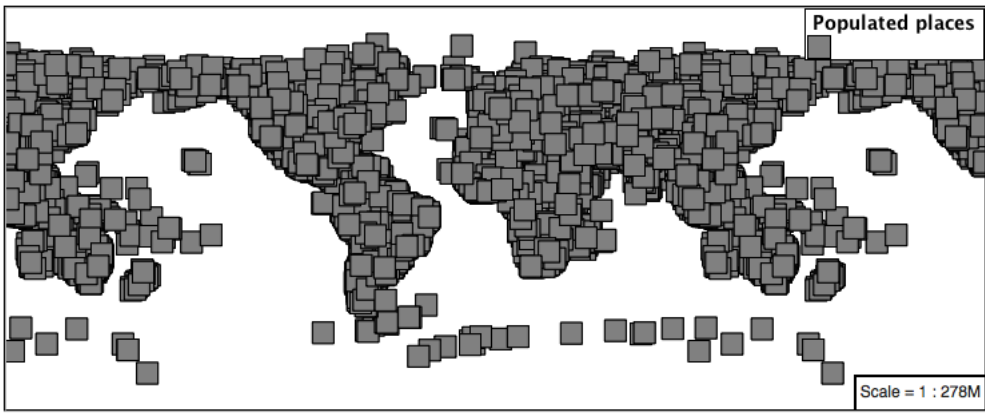
Points are represented with the mandatory property **mark**.

The SLD standard provides "well-known" symbols for use with point symbology: `circle`, `square`, `triangle`, `arrow`, `cross`, `star`, and `x`.

1. As a **key property** the presence **mark** triggers the generation of an appropriate PointSymbolizer.

```
* {
 mark: symbol(square);
}
```

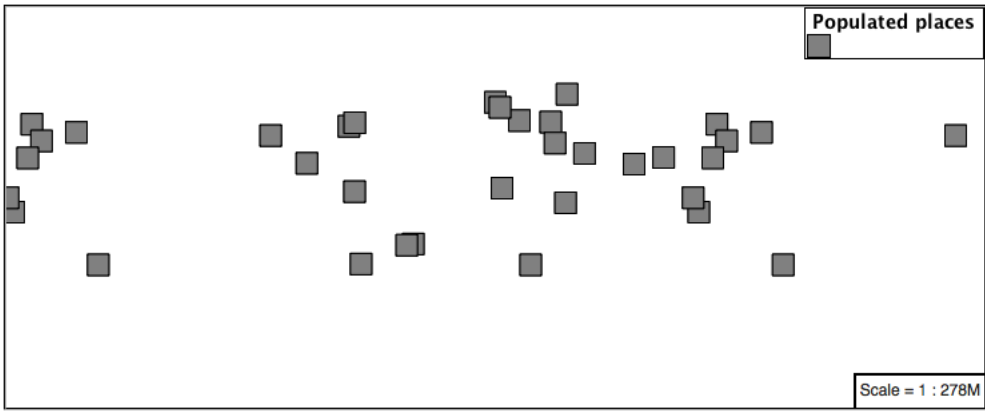
2. Map Preview:



3. Before we continue we will use a selector to cut down the amount of data shown to a reasonable level.

```
[ SCALERANK < 1 ] {
  mark: symbol(square);
}
```

4. Resulting in a considerably cleaner image:



5. Additional properties are available to control a mark's presentation:

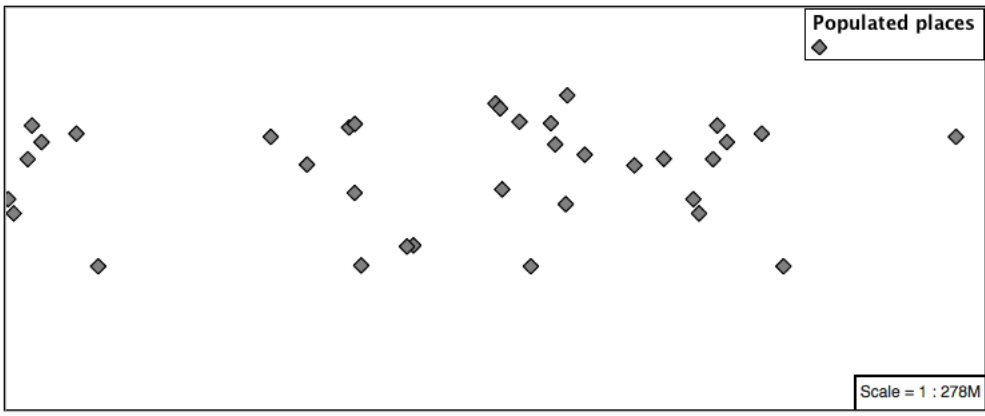
   The **mark-size** property is used to control symbol size.

   The **mark-rotation** property controls orientation, accepting input in degrees.

   Trying these two settings together:

```
[ SCALERANK < 1 ] {
  mark: symbol(square);
  mark-size: 8;
  mark-rotation: 45;
}
```

6. Results in each location being marked with a diamond:



7. Now that we have assigned our point location a symbol we can make use of a **pseudo-selector** to style the resulting shape.

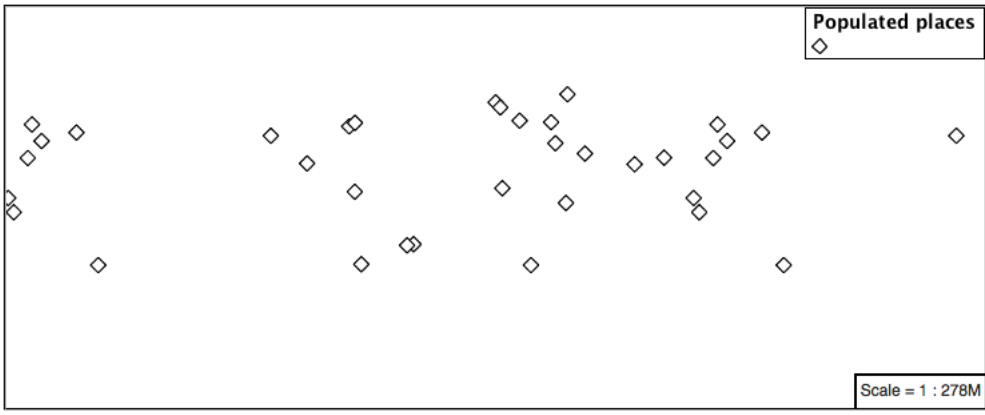
   :symbol - provides styling for all the symbols in the CSS document.

   :mark - provides styling for all the mark symbols in the CSS document.

This form of pseudo-selector is used for all marks:

```
[ SCALERANK < 1 ] {
  mark: symbol(square);
  mark-size: 8;
  mark-rotation: 45;
}
:mark{
   fill: white;
   stroke: black;
}
```

8. Updating the mark to a white square with a black outline.



9. The second approach is used to individual configure symbols in the same document.

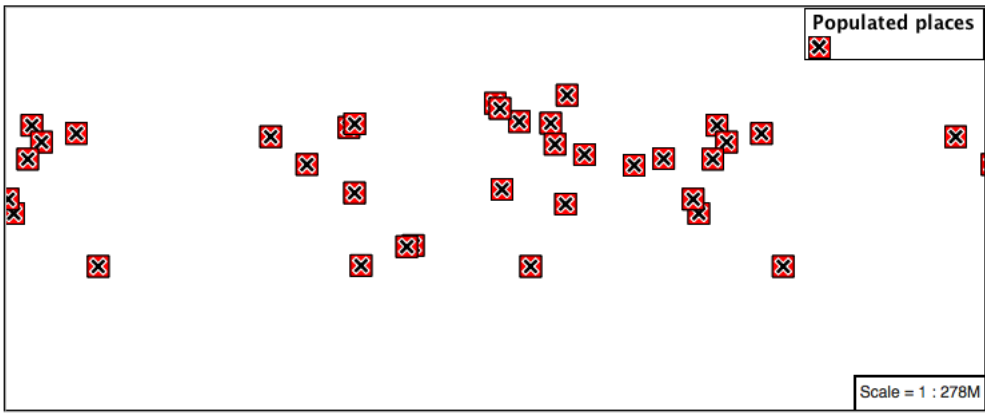
   :nth-symbol(1) - if needed we could specify which symbol in the document we wish to modify.

   :nth-mark(1) - provides styling for the first mark symbol in the CSS document.

   Using this approach marks can be composed of multiple symbols, each with its own settings:

```
[ SCALERANK < 1 ] {
  mark: symbol(square),symbol(cross);
  mark-size: 16,14;
  mark-rotation: 0,45;
}
:nth-mark(1){
   fill: red;
   stroke: black;
}
:nth-mark(2){
   fill: black;
   stroke: white;
}
```

10. Producing an interesting compound symbol effect:



Graphic

Symbols can also be supplied by an external graphic,

This technique was shown with the initial file:*airport.svg* CSS example.

1. To use an external graphic two pieces of information are required.

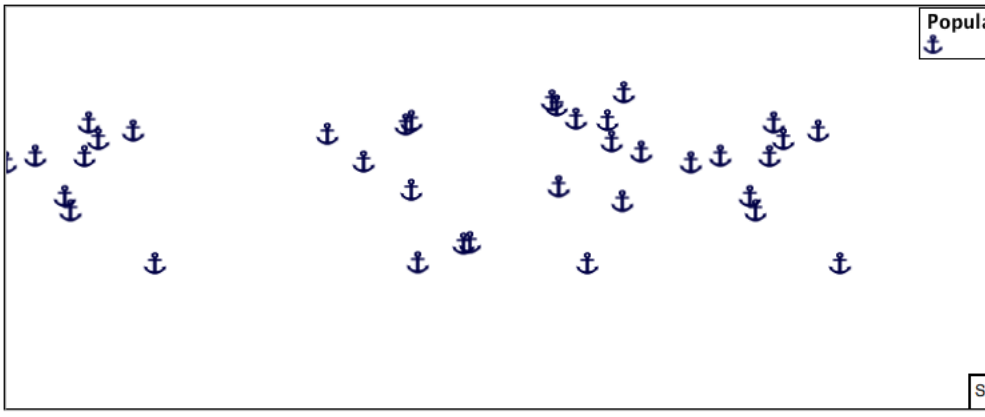
   mark property is defined with a **url** reference to image.

   mark-mime property is used to tell the rendering engine what file format to expect

   This technique is used to reference files placed in the styles directory.

```
[ SCALERANK < 1 ] {
  mark: url(port.svg);
  mark-mime: "image/svg";
}
```

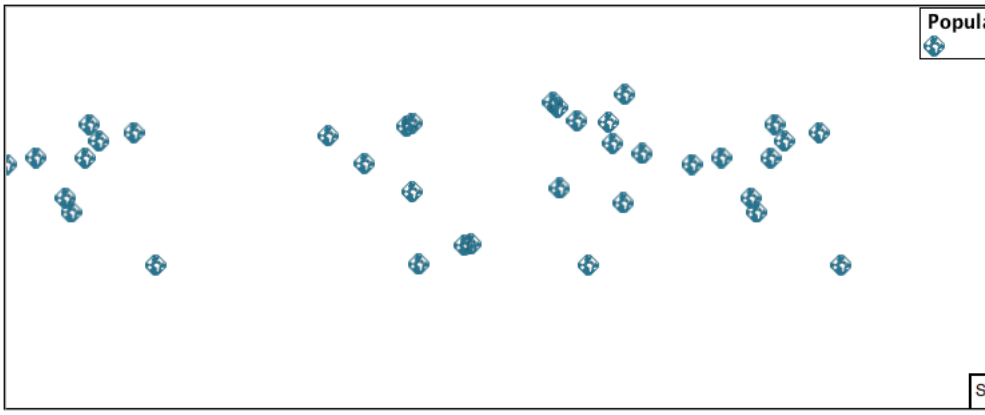
2. Drawing the provided shape in each location:



3. The **mark** property **url** reference can also be used to reference external images. We can make use of the GeoServer logo.

```
[ SCALERANK < 1 ] {
    mark: url(
↪"http://localhost:8080/geoserver/web/wicket/resource/
↪org.geoserver.web.GeoServerBasePage/img/logo.png");
    mark-mime: "image/png";
    mark-size: 16;
}
```

4. As shown in the map preview.



Label

Labeling is now familiar from our experience with LineString and Polygons.

The key properties **mark** and **label** are required to label Point locations.

1. Replace `point_example` with the following:

```
[ SCALERANK < 1 ] {
  mark: symbol(circle);
  label: [NAME];
}
```

2. Confirm the result in `Map` preview.

3. Each label is drawn starting from the provided point - which is unfortunate as it assures each label will overlap with the symbol used. To fix this limitation we will make use of the SLD controls for label placement:

   label-anchor provides two values expressing how a label is aligned with respect to the starting label position.

   label-offset is be used to provide an initial displacement using and x and y offset. For points this offset is recommended to adjust the label position away for the area used by the symbol.

   Note: The property **label-anchor** defines an anchor position relative to the bounding box formed by the resulting label. This anchor position is snapped to the label position generated by the point location and displacement offset.

4. Using these two facilities together we can center our labels below the symbol, taking care that the displacement used provides an offset just outside the area required for the symbol size.

```
[ SCALERANK < 1 ] {
  mark: symbol(circle);
  mark-size: 10;

  label: [NAME];
  label-offset: 0 -12;
  label-anchor: 0.5 1.0;

  font-fill: black;
}
```

5. Each label is now placed under the mark.

6. One remaining issue is the overlap between labels and symbols.

   GeoServer provides a vendor specific parameter to allow symbols to take part in label conflict resolution, preventing labels from overlapping any symbols. This severely limits the area available for labeling and is best used in conjunction with a large maximum displacement vendor option.

   mark-label-obstacle vendor parameter asks the rendering engine to avoid drawing labels over top of the indicated symbol.

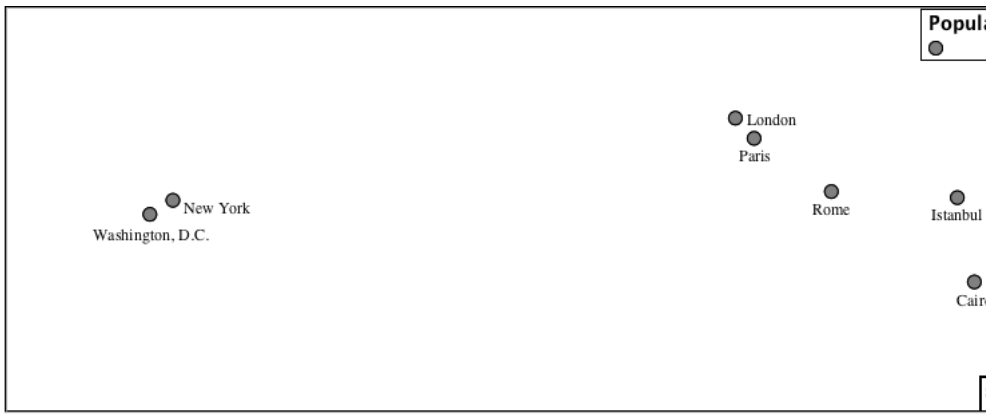
   label-max-displacement vendor parameter provides the rendering engine a maximum distance it is allowed to move labels during conflict resolution.

   label-padding vendor parameter tells the rendering engine to provide a minimum distance between the labels on the map, ensuring they do not overlap.

   Update our example to use these settings:

```
[ SCALERANK < 1 ] {
  mark: symbol(circle);
  mark-size: 10;

  label: [NAME];
  label-offset: 0 -12;
  label-anchor: 0.5 1.0;

  font-fill: black;

  mark-label-obstacle: true;
  label-max-displacement: 100;
  label-padding: 2;
}
```

7. Resulting in a considerably cleaner image:

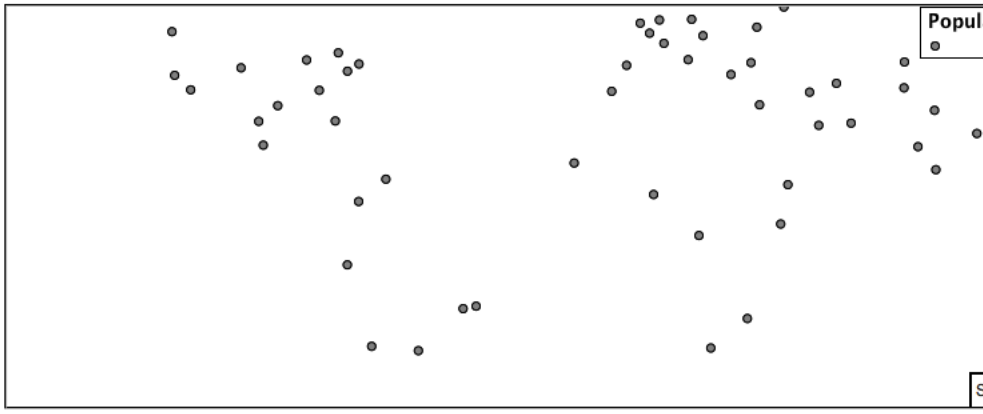


Dynamic Styling

1. We will quickly use **scalerank** to select content based on @scale selectors.

```
[@scale < 4000000] {
   mark: symbol(circle);
}
[@scale > 4000000] [@scale < 8000000] [SCALERANK < 7] {
   mark: symbol(circle);
}

[@scale␣
↪> 8000000] [@scale < 17000000] [SCALERANK < 5] {
   mark: symbol(circle);
}

[@scale␣
↪> 17000000] [@scale < 35000000] [SCALERANK < 4] {
   mark: symbol(circle);
}

[@scale␣
↪> 35000000] [@scale < 70000000][SCALERANK < 3] {
   mark: symbol(circle);
}

[@scale␣
↪> 70000000] [@scale < 140000000][SCALERANK < 2] {
   mark: symbol(circle);
}

[@scale > 140000000] [SCALERANK < 1] {
  mark: symbol(circle);
}

* {
  mark-size: 6;
}
```

2. Click *Submit* to update the *Map* after each step.



3. To add labeling we must use both the **key properties** mark and label in each scale selector, using rule cascading to define the mark-size and font information once.

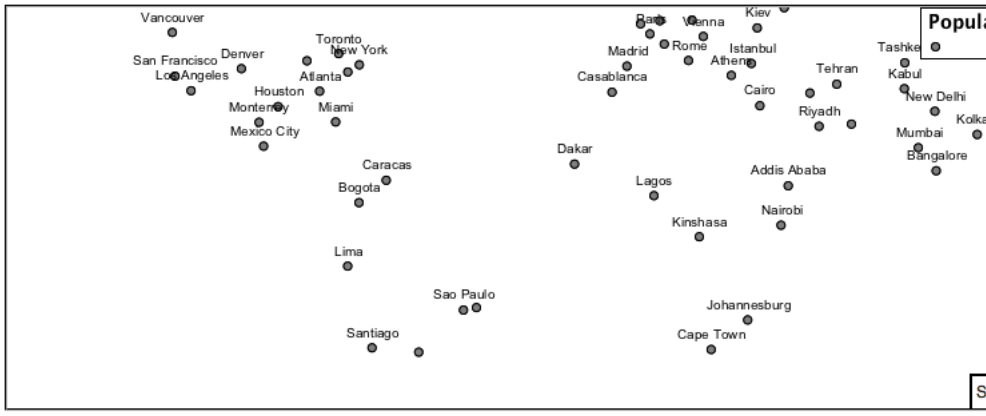
```
[@scale < 4000000] {
   mark: symbol(circle);
   label: [NAME];
}
[@scale > 4000000] [@scale < 8000000] [SCALERANK < 7] {
   mark: symbol(circle);
   label: [NAME];
}

[@scale␣
↪> 8000000] [@scale < 17000000] [SCALERANK < 5] {
   mark: symbol(circle);
   label: [NAME];
}

[@scale␣
↪> 17000000] [@scale < 35000000] [SCALERANK < 4] {
   mark: symbol(circle);
   label: [NAME];
}

[@scale␣
↪> 35000000] [@scale < 70000000][SCALERANK < 3] {
   mark: symbol(circle);
   label: [NAME];
}

[@scale␣
↪> 70000000] [@scale < 140000000][SCALERANK < 2] {
   mark: symbol(circle);
   label: [NAME];
}

[@scale > 140000000] [SCALERANK < 1] {
   mark: symbol(circle);
   label: [NAME];
}

* {
  mark-size: 6;
```

```
  font-fill: black;
  font-family: "Arial";
  font-size: 10;
}
```

4. We will use **label-offset** and **label-anchor** to position the label above each symbol.

   Add the following two lines to the * selector:

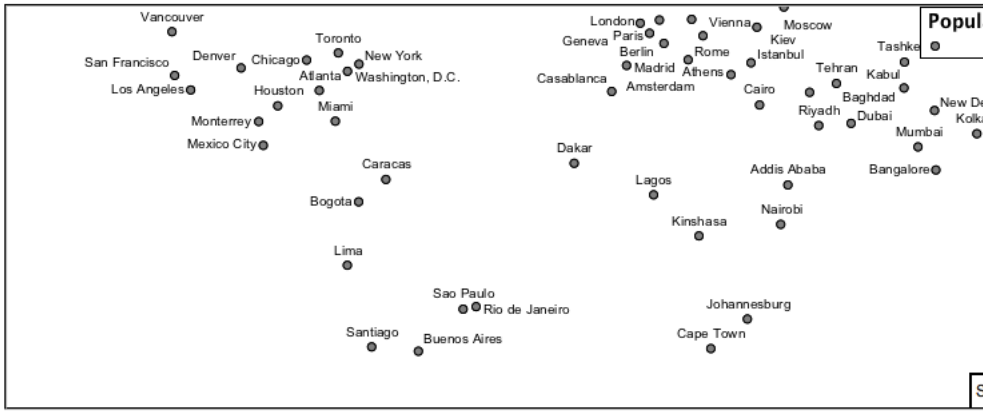
```
* {
  mark-size: 6;

  font-fill: black;
  font-family: "Arial";
  font-size: 10;

  label-anchor: 0.5 0;
  label-offset: 0 6;
}
```



5. A little bit of work with vendor specific parameters will prevent our labels from colliding with each symbol, while giving the rendering engine some flexibility in how far it is allowed to relocate a label.

   Add the following vendor options to the * selector:

```
* {
  mark-size: 6;

  font-fill: black;
  font-family: "Arial";
  font-size: 10;

  label-anchor: 0.5 0;
  label-offset: 0 6;

  mark-label-obstacle: true;
  label-max-displacement: 90;
  label-padding: 2;
}
```



6. Now that we have clearly labeled our cities, zoom into an area you are familiar with and we can look at changing symbology on a case-by-case basis.

   We have used expressions previous to generate an appropriate label. Expressions can also be used for many other property settings.

   The `ne:populated_places` layer provides several attributes specifically to make styling easier:

- **SCALERANK**: we have already used this attribute to control the level of detail displayed
- **LABELRANK**: hint used for conflict resolution, allowing important cities such as capitals to be labeled even when they are close to a larger neighbor.
- **FEATURECLA**: used to indicate different types of cities. We will check for `Admin-0 capital` cities.

  The first thing we will do is calculate the **mark-size** using a quick expression:

```
[10-(SCALERANK/2)]
```

This expression should result in sizes between 5 and 9 and will need to be applied to both **mark-size** and **label-offset**.

Rather than the "first come first served" default to resolve labeling conflicts we can manually provide GeoServer with a label priority.

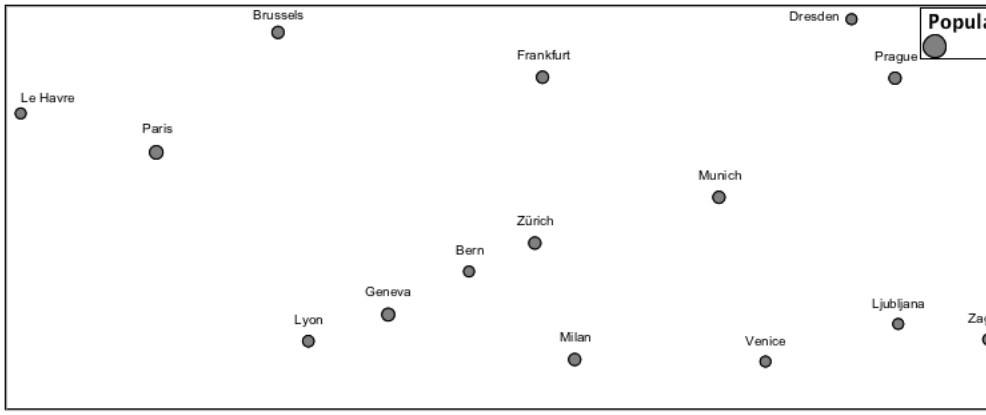
The expression provided is calculated for each label, in the event of a conflict the label with the highest priority takes precedence.

The LABELRANK attribute goes from 1 through 10 and needs to be flipped around before use as a GeoServer label priority:

```
[10 - LABELRANK]
```

This expression will result in values between 0 and 10 and will be used for the **label-priority**.

```
* {
  mark-size: [10-(SCALERANK/2)];

  font-fill: black;
  font-family: "Arial";
  font-size: 10;

  label-anchor: 0.5 0;
  label-offset: 0 [10-(SCALERANK/2)];

  mark-label-obstacle: true;
  label-max-displacement: 90;
  label-padding: 2;
  label-priority: [10 - LABELRANK];
}
```



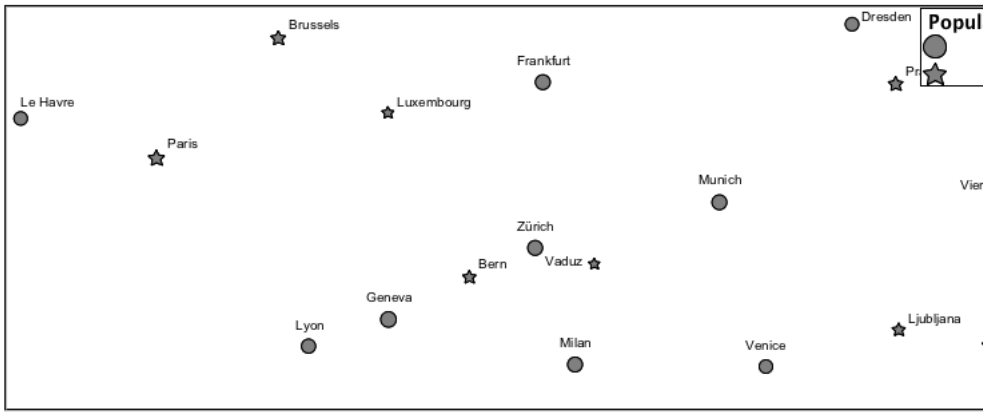
7. Next we can use `FEATURECLA` to check for capital cities.

   Adding a selector for capital cities at the top of the file:

```
/* capitals */
[@scale < 70000000]
[FEATURECLA = 'Admin-0 capital']  {
   mark: symbol(star);
   label: [NAME];
}
[@scale > 70000000] [SCALERANK < 2]
[FEATURECLA = 'Admin-0 capital']  {
   mark: symbol(star);
   label: [NAME];
}
```

And updating the populated places selectors to ignore capital cities:

```
/* populated places */
[@scale < 4000000]
[FEATURECLA <> 'Admin-0 capital'] {
   mark: symbol(circle);
   label: [NAME];
}
[@scale > 4000000] [@scale < 8000000] [SCALERANK < 7]
[FEATURECLA <> 'Admin-0 capital'] {
   mark: symbol(circle);
   label: [NAME];
}

[@scale > 8000000] [@scale < 17000000] [SCALERANK < 5]
[FEATURECLA <> 'Admin-0 capital'] {
   mark: symbol(circle);
   label: [NAME];
}

[@scale > 17000000] [@scale < 35000000] [SCALERANK < 4]
[FEATURECLA <> 'Admin-0 capital'] {
   mark: symbol(circle);
   label: [NAME];
}

[@scale > 35000000] [@scale < 70000000][SCALERANK < 3]
[FEATURECLA <> 'Admin-0 capital'] {
   mark: symbol(circle);
   label: [NAME];
}

[@scale > 70000000] [@scale < 140000000][SCALERANK < 2]
[FEATURECLA <> 'Admin-0 capital'] {
   mark: symbol(circle);
   label: [NAME];
}

[@scale > 140000000] [SCALERANK < 1]
[FEATURECLA <> 'Admin-0 capital'] {
   mark: symbol(circle);
   label: [NAME];
}
```



8. Finally we can fill in the capital city symbols using a combination

of a selector to detect capital cities, and pseudo selector to provide mark styling.

```
[FEATURECLA = 'Admin-0 capital'] :mark {
  fill: black;
}

:symbol {
  fill: gray;
  stroke: black;
}
```

9. If you would like to check your work the final file is here: `point_example.css`

Bonus

Challenge Geometry Location

1. The **mark** property can be used to render any geometry content.
2. **Challenge:** Try this yourself by rendering a polygon layer using a **mark** property.

Note: Answer *discussed* at the end of the workbook.

Explore Dynamic Symbolization

1. We went to a lot of work to set up selectors to choose between symbol(star) and symbol(circle) for capital cities.

   This approach is straightforward when applied in isolation:

```
[FEATURECLA = 'Admin-0 capital'] {
   mark: symbol(star);
}
[FEATURECLA <> 'Admin-0 capital'] {
   mark: symbol(circle);
}
```

When combined with checking another attribute, or checking @scale as in our example, this approach can quickly lead to many rules which can be difficult to keep straight.

2. Taking a closer look both `symbol()` and `url()` can actually be expressed using a string:

```
[FEATURECLA = 'Admin-0 capital'] {
   mark: symbol("star");
}
```

Which is represented in SLD as:

```
<sld:PointSymbolizer>
  <sld:Graphic>
     <sld:Mark>
        <sld:WellKnownName>star</sld:WellKnownName>
        <sld:Fill/>
        <sld:Stroke/>
     </sld:Mark>
  </sld:Graphic>
</sld:PointSymbolizer>
```

3. GeoServer recognizes this limitation of SLD Mark and ExternalGraphic and provides an opportunity for dynamic symbolization.

   This is accomplished by embedding a small CQL expression in the string passed to symbol or url. This sub-expression is isolated with `${ }` as shown:

```
* {
   mark: symbol(
     "${if_then_else(equalTo(FEATURECLA,
↪'Admin-0 capital'),'star','circle')}"
   );
}
```

Which is represented in SLD as:

```
<sld:PointSymbolizer>
  <sld:Graphic>
     <sld:Mark>
        <sld:WellKnownName>
↪${if_then_else(equalTo(FEATURECLA,'Admin-
↪0 capital'),'star','circle')}</sld:WellKnownName>
        <sld:Fill/>
        <sld:Stroke/>
     </sld:Mark>
  </sld:Graphic>
</sld:PointSymbolizer>
```

4. **Challenge:** Use this approach to rewrite the *Dynamic Styling* example.

Note: Answer *provided* at the end of the workbook.

Challenge Layer Group

1. Use a **Layer Group** to explore how symbology works together to form a map.

- ne:NE1
- ne:states_provincces_shp
- ne: populated_places

2. To help start things out here is a style for `ne:states_provinces_shp`:

```
* {
   fill: white,[
    recode(mapcolor9,
      1,'#8dd3c7', 2,'#ffffb3', 3,'#bebada',
      4,'#fb8072', 5,'#80b1d3', 6,'#fdb462',
      7,'#b3de69', 8,'#fccde5', 9,'#d9d9d9')
   ];
   fill-opacity: 05%,50%;

   stroke: black;
   stroke-width: 0.25;
   stroke-opacity: 50%;
}
```

3. This background is relatively busy and care must be taken to ensure both symbols and labels are clearly visible.

4. **Challenge:** Do your best to style populated_places over this busy background.

   Here is an example with labels for inspiration:

Note: Answer *provided* at the end of the workbook.

Explore True Type Fonts

1. In addition to image formats GeoServer can make use other kinds of graphics, such as True Type fonts:

```
* {
   mark: symbol("ttf://Webdings#0x0064");
}
:mark {
   stroke: blue;
}
```

2. Additional fonts dropped in the `styles` directory are available for use.

Explore Custom Graphics

1. The GeoServer rendering engine allows Java developers to hook in additional symbol support.

   This facility is used by GeoServer to offer the shapes used for pattern fills. Community extensions allow the use of simple custom shapes and even charts.

2. Support has been added for custom grpahics using the WKT Geometry representation.

```
* {
   mark: symbol("wkt://MULTILINESTRING((-0.25␣
↪-0.25, -0.125 -0.25), (0.125 -0.25, 0.25 -0.25), (-
↪0.25 0.25, -0.125 0.25), (0.125 0.25, 0.25 0.25))");
}
:mark {
   stroke: blue;
}
```

Rasters

Finally we will look at using CSS styling for the portrayal of raster data.

Fig. 6.299: Raster Symbology

Review of raster symbology:

- Raster data is **Grid Coverage** where values have been recorded in a regular array. In OGC terms a **Coverage** can be used to look up a value or measurement for each location.
- When queried with a "sample" location:
  - A grid coverage can determine the appropriate array location and retrieve a value. Different techniques may be used interpolate an

appropriate value from several measurements (higher quality) or directly return the "nearest neighbor" (faster).

- A vector coverages would use a point-in-polygon check and return an appropriate attribute value.
- A scientific model can calculate a value for each sample location

- Many raster formats organize information into bands of content. Values recorded in these bands and may be mapped into colors for display (a process similar to theming an attribute for vector data).

  For imagery the raster data is already formed into red, green and blue bands for display.

- As raster data has no inherent shape, the format is responsible for describing the orientation and location of the grid used to record measurements.

  These raster examples use a digital elevation model consisting of a single band of height measurements. The imagery examples use an RGB image that has been hand coloured for use as a base map.

  Reference:

- *Raster Symbology* (User Manual | CSS Property Listing)
- *Rasters* (User Manual | CSS Cookbook);
- *Point* (User Manual | SLD Reference)

The exercise makes use of the `usgs:dem` and `ne:ne1` layers.

Image

The **raster-channels** is the **key property** for display of images and raster data. The value `auto` is recommended, allowing the image format to select the appropriate red, green and blue channels for display.

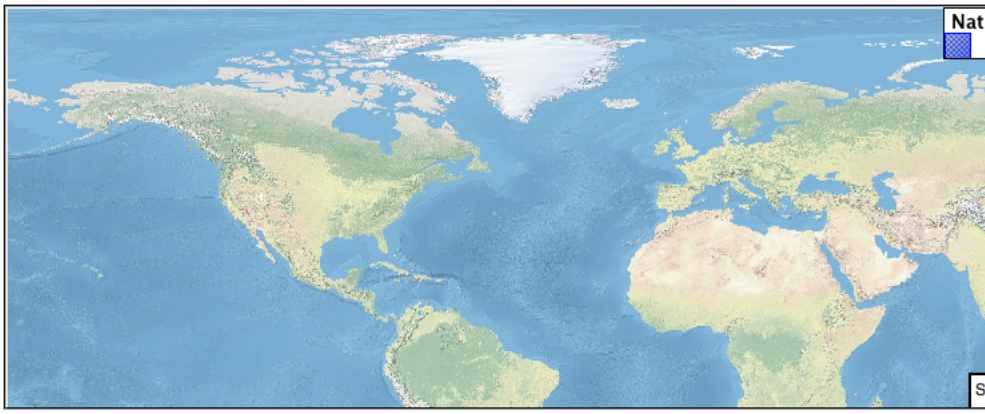
1. Navigate to the **Styles** page.
2. Click *Add a new style* and choose the following:

Name:	`image_example`
Workspace:	`No workspace`
Format:	`CSS`

3. Replace the initial CSS definition with:

```
* {
  raster-channels: auto;
}
```

4. And use the *Layer Preview* tab to preview the result.



5. If required a list three band numbers can be supplied (for images recording in several wave lengths) or a single band number can be used to view a grayscale image.

```
* {
  raster-channels: 2;
}
```

6. Isolating just the green band (it wil be drawn as a grayscale image):

DEM

A digital elevation model is an example of raster data made up of measurements, rather than colors information.

The `usgs:dem` layer used used for this exercise:

1. Return to the the **Styles** page.
2. Click *Add a new style* and choose the following:

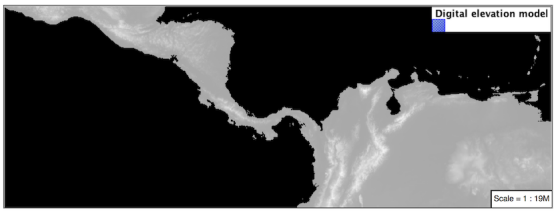
Name:	`raster_example`
Workspace:	`No workspace`
Format:	`CSS`

3. When we use the **raster-channels** property set to `auto` the rendering engine will select our single band of raster content, and do its

best to map these values into a grayscale image. Replace the content of the style with:

```
* {
  raster-channels: auto;
}
```

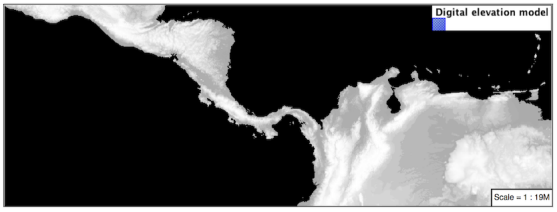
4. Use the *Layer Preview* tab to preview the result. The range produced in this case from the highest and lowest values.



5. We can use a bit of image processing to emphasis the generated color mapping by making use **raster-contrast-enhancement**.

```
* {
  raster-channels: 1;
  raster-contrast-enhancement: histogram;
}
```

6. Image processing of this sort should be used with caution as it does distort the presentation (in this case making the landscape look more varied then it is in reality.



Color Map

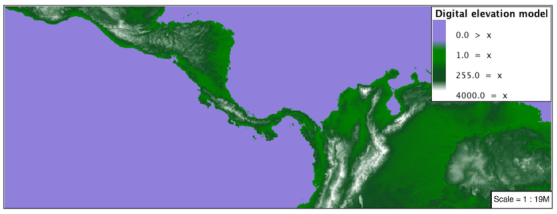
The approach of mapping a data channel directly to a color channel is only suitable to quickly look at quantitative data.

For qualitative data (such as land use) or simply to use color, we need a different approach:

1. Apply the following CSS to our *usgs:DEM* layer:

```
* {
  raster-channels: auto;
  raster-color-map: color-map-entry(#9080DB, 0)
                    color-map-entry(#008000, 1)
                    color-map-entry(#105020, 255)
                    color-map-entry(#FFFFFF, 4000);
}
```

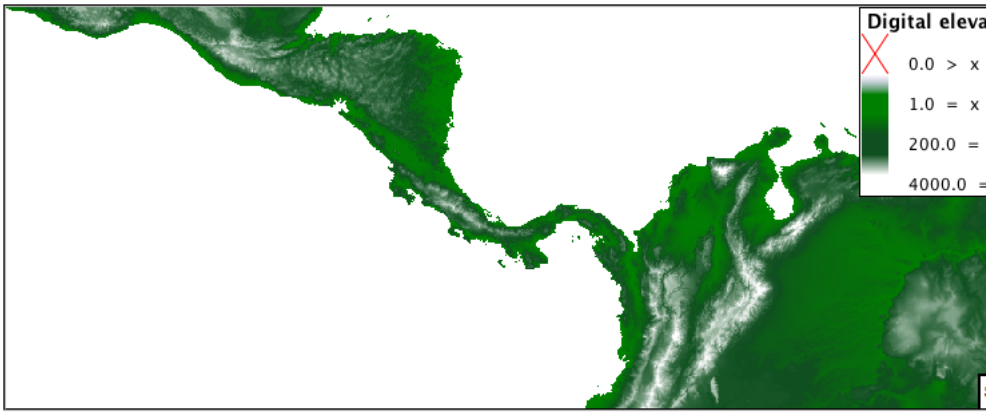
2. Resulting in this artificial color image:



3. An opacity value can also be used with **color-map-entry**.

```
* {
  raster-channels: auto;
  raster-color-map: color-map-entry(#9080DB, 0, 0.0)
                    color-map-entry(#008000, 1, 1.0)
                    color-map-entry(#105020, 200, 1.0)
↪                   color-map-entry(#FFFFFF, 4000, 1.0);
}
```

4. Allowing the areas of zero height to be transparent:



5. Raster format for GIS work often supply a "no data" value, or contain a mask, limiting the dataset to only the locations with valid information.

Custom

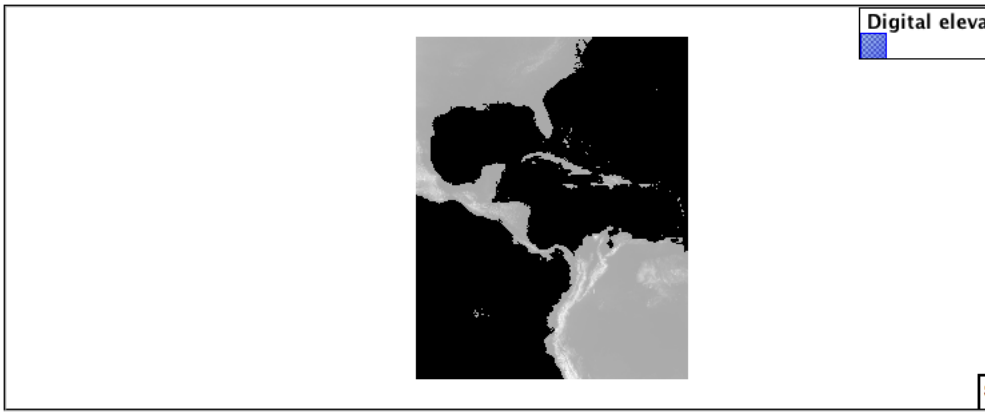
We can use what we have learned about color maps to apply a color brewer palette to our data.

This exploration focuses on accurately communicating differences in value, rather than strictly making a pretty picture. Care should be taken to consider the target audience and medium used during palette selection.

1. Restore the `raster_example` CSS style to the following:

```
* {
  raster-channels: auto;
}
```

2. Producing the following map preview.

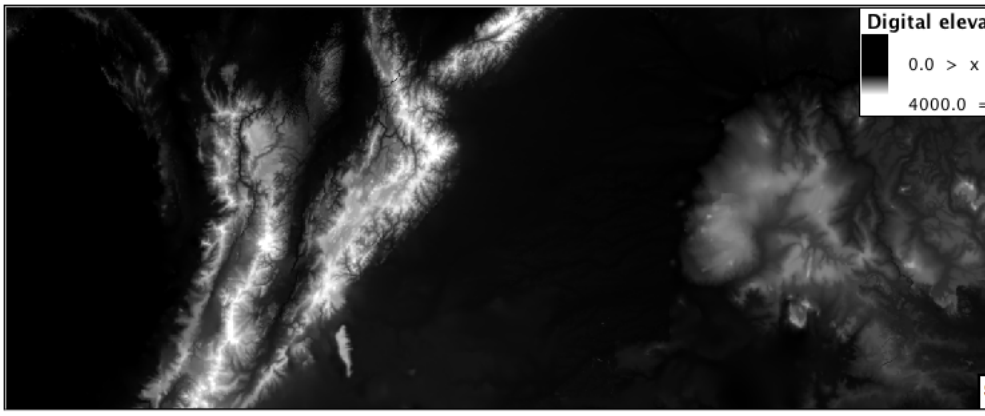


3. To start with we can provide our own grayscale using two color map entries.

```
* {
  raster-channels: auto;
  raster-color-map: color-map-entry(#000000, 0)
                    color-map-entry(#FFFFFF, 4000);
}
```

4. Use the *Map* tab to zoom in and take a look.

   This is much more direct representation of the source data. We have used our knowledge of elevations to construct a more accurate style.



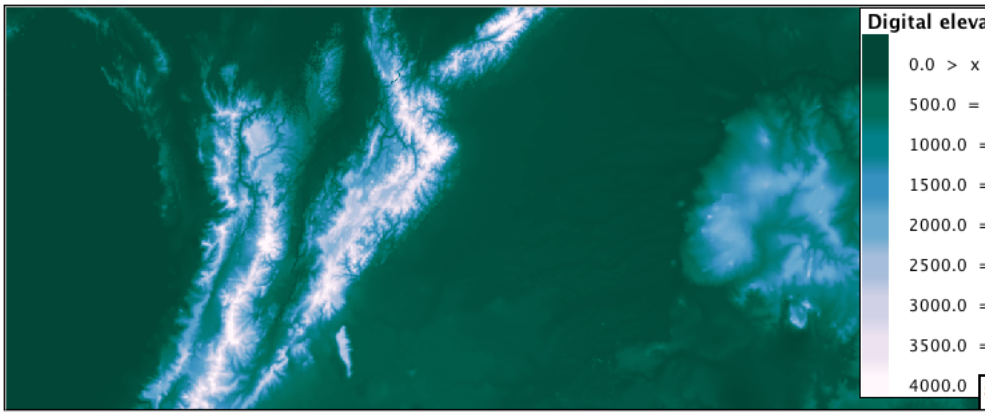
5. While our straightforward style is easy to understand, it does leave a bit to be desired with respect to clarity.

   The eye has a hard time telling apart dark shades of black (or bright shades of white) and will struggle to make sense of this image. To address this limitation we are going to switch to the ColorBrewer **9-class PuBuGn** palette. This is a sequential palette that has been hand tuned to communicate a steady change of values.

6. Update your style with the following:

```
* {
  raster-channels: auto;
```

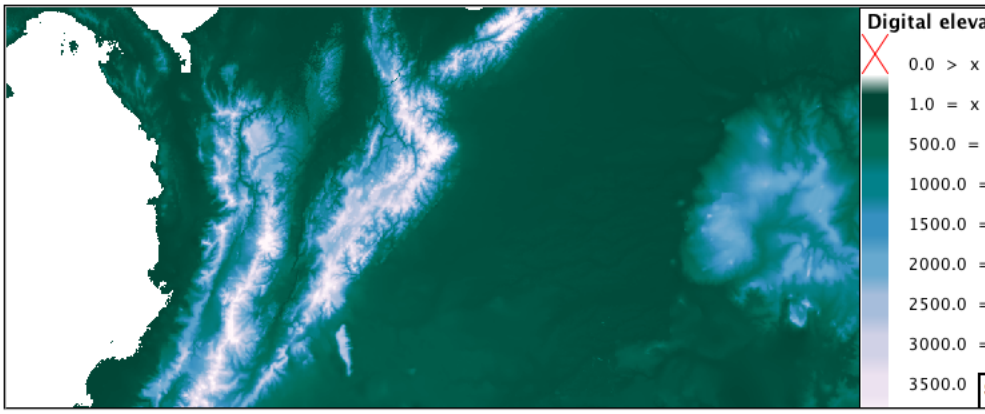
```
                raster-color-map:
                   color-map-entry(#014636,   0)
                   color-map-entry(#016c59, 500)
                   color-map-entry(#02818a,1000)
                   color-map-entry(#3690c0,1500)
                   color-map-entry(#67a9cf,2000)
                   color-map-entry(#a6bddb,2500)
                   color-map-entry(#d0d1e6,3000)
                   color-map-entry(#ece2f0,3500)
                   color-map-entry(#fff7fb,4000);
             }
```



7. A little bit of work with alpha (to mark the ocean as a no-data section):

```
             * {
               raster-channels: auto;
               raster-color-map:
                   color-map-entry(#014636,   0,0)
                   color-map-entry(#014636,   1)
                   color-map-entry(#016c59, 500)
                   color-map-entry(#02818a,1000)
                   color-map-entry(#3690c0,1500)
                   color-map-entry(#67a9cf,2000)
                   color-map-entry(#a6bddb,2500)
                   color-map-entry(#d0d1e6,3000)
                   color-map-entry(#ece2f0,3500)
                   color-map-entry(#fff7fb,4000);
             }
```

8. And we are done:



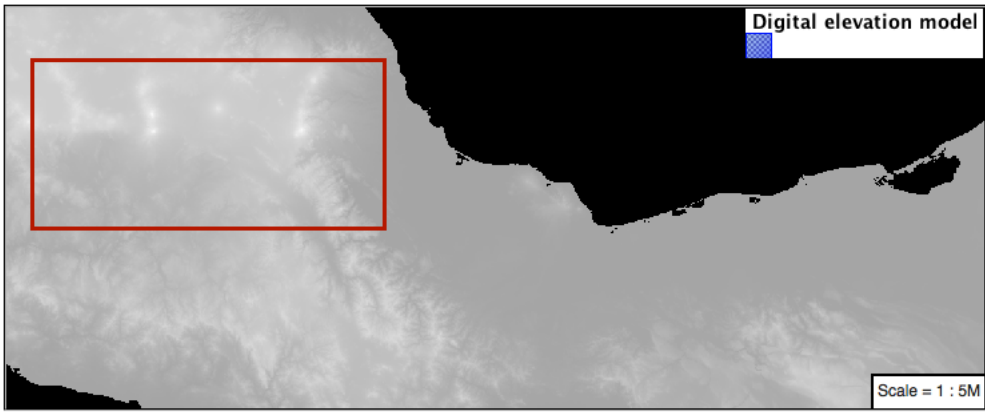
Bonus

Explore Contrast Enhancement

1. A special effect that is effective with grayscale information is automatic contrast adjustment.
2. Make use of a simple contrast enhancement with `usgs:dem`:

```
* {
    raster-channels: auto;
    raster-contrast-enhancement: normalize;
}
```

3. Can you explain what happens when zoom in to only show a land area (as indicated with the bounding box below)?



Note: Discussion *provided* at the end of the workbook.

Challenge Intervals

1. The **raster-color-map-type** property dictates how the values are used to generate a resulting color.

- `ramp` is used for quantitative data, providing a smooth interpolation between the provided color values.
- `intervals` provides categorization for quantitative data, assigning each range of values a solid color.
- `values` is used for qualitative data, each value is required to have a **color-map-entry** or it will not be displayed.

2. **Chalenge:** Update your DEM example to use **intervals** for presentation. What are the advantages of using this approach for elevation data?

Note: Answer *provided* at the end of the workbook.

Explore Image Processing

Additional properties are available to provide slight image processing during visualization.

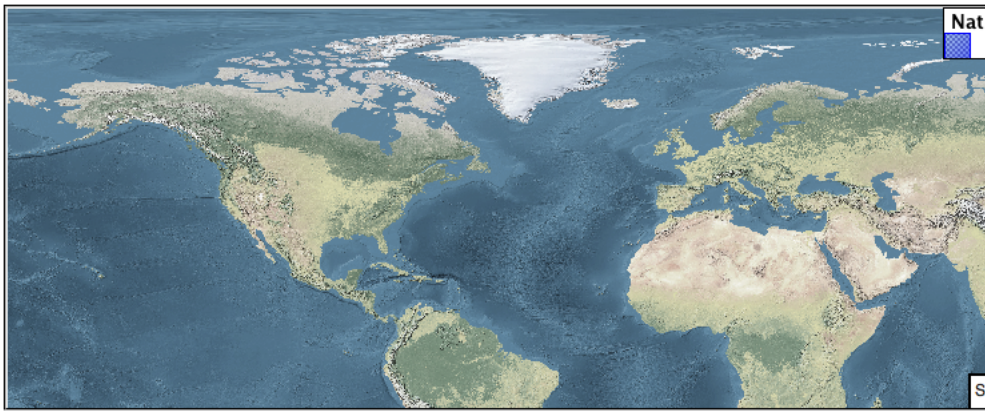
Note: In this section are we going to be working around a preview issue where only the top left corner of the raster remains visible during image processing. This issue has been reported as GEOS-6213.

Image processing can be used to enhance the output to highlight small details or to balance images from different sensors allowing them to be compared.

1. The **raster-contrast-enhancement** property is used to turn on a range of post processing effects. Settings are provided for `normalize` or `histogram` or `none`;

```
* {
    raster-channels: auto;
    raster-contrast-enhancement: normalize;
}
```

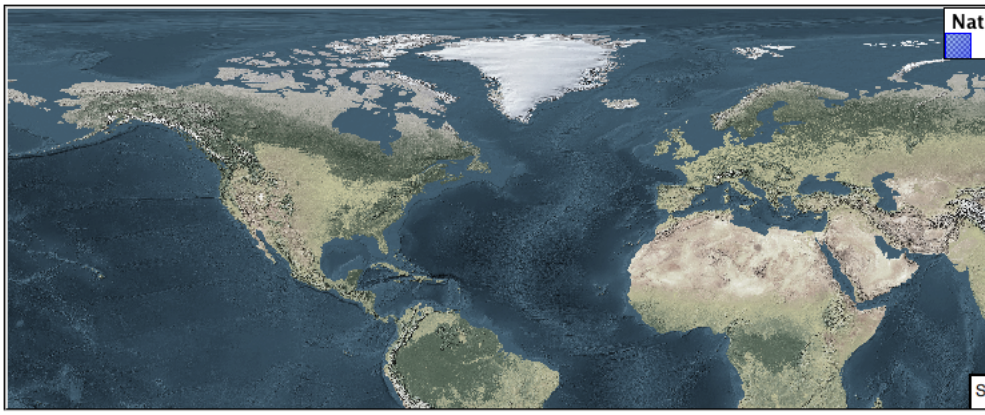
2. Producing the following image:



3. The **raster-gamma** property is used adjust the brightness of **raster-contrast-enhancement** output. Values less than 1 are used to brighten the image while values greater than 1 darken the image.

```
* {
   raster-channels: auto;
   raster-contrast-enhancement: none;
   raster-gamma: 1.5;
}
```

4. Providing the following effect:



Challenge Clear Digital Elevation Model Presentation

1. Now that you have seen the data on screen and have a better understanding how would you modify our initial gray-scale example?
2. **Challenge:** Use what you have learned to present the `usgs:dem` clearly.

Note: Answer *provided* at the end of the workbook.

Challenge Raster Opacity

1. There is a quick way to make raster data transparent, **raster-opacity** property works in the same fashion as with vector data. The raster as a whole will be drawn partially transparent allow content from other layers to provide context.
2. **Challenge:** Can you think of an example where this would be useful?

Note: Discussion *provided* at the end of the workbook.

CSS Workbook Conclusion

We hope you have enjoyed this styling workshop.

Additional resources:

- *CSS Extension*
- *CSS Cookbook*

CSS Workbook Answer Key

The following questions were listed through out the workshop as an opportunity to explore the material in greater depth. Please do your best to consider the questions in detail prior to checking here for the answer. Questions are provided to teach valuable skills, such as a chance to understand how feature type styles are used to control z-order, or where to locate information in the user manual.

SLD Generation

Answer for *Challenge SLD Generation*:

1. Generate the SLD for the following CSS.

```
* {
  stroke: black;
}
```

2. **Challenge:** What is unusual about the generated SLD? Can you explain why it still works as expected?

   The generated SLD does not contain any stroke properties, even though black was specified:

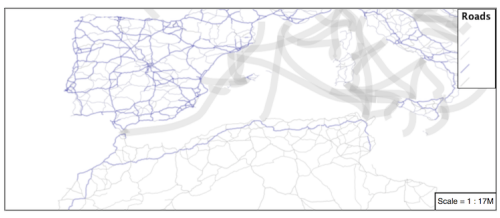
```
<sld:LineSymbolizer>
  <sld:Stroke/>
</sld:LineSymbolizer>
```

   SLD considers black the default stroke color for a LineSymbolizer, so no further detail was required.

Classification

Answer for *Challenge Classification*:

1. **Challenge:** Create a new style adjust road appearance based on **type**.



Hint: The available values are 'Major Highway','Secondary Highway','Road' and 'Unknown'.

2. Here is an example:

```
[type = 'Major Highway' ] {
    stroke: #000088;
    stroke-width: 1.25;
}
[type = 'Secondary Highway' ]{
    stroke: #8888AA;
    stroke-width: 0.75;
}
[type = 'Road']{
    stroke: #888888;
    stroke-width: .75;
}
[type = 'Unknown' ]{
    stroke: #888888;
    stroke-width: 0.5;
}
* {
   stroke: #AAAAAA;
   stroke-opacity: 0.25;
   stroke-width: 0.5;
}
```

SLD Z-Index Generation

Answer for *Challenge SLD Z-Index Generation*:

1. Review the SLD generated by the **z-index** example.

```
* {
  stroke: black, #8080E6;
  stroke-width: 5px, 3px;
  z-index: 0, 1;
}
```

2. *Challenge:* There is an interesting trick in the generated SLD, can you explain how it works?

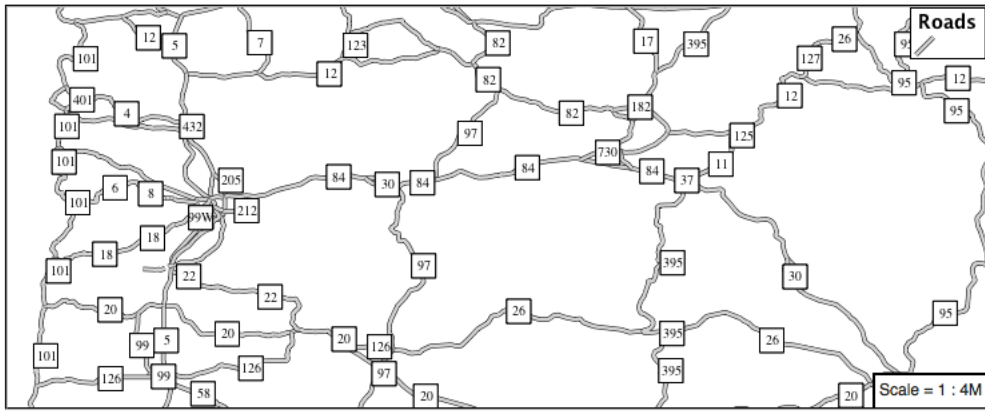
3. The Z-Order example produces multiple FeatureTypeSytle definitions, each acting like an "inner layer".

   Each FeatureTypeStyle is rendered into its own raster, and the results merged in order. The legend shown in the map preview also provides a hint, as the rule from each FeatureType style is shown.

Label Shields

Answer for *Challenge Label Shields*:

1. The traditional presentation of roads in the US is the use of a shield symbol, with the road number marked on top.



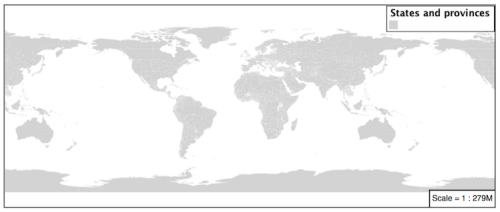
1. *Challenge:* Have a look at the documentation and reproduce this technique.
2. The use of a label shield is a vendor specific capability of the GeoServer rendering engine. The tricky part of this exercise is finding the documentation online (i.e. *Styled Marks in CSS*).

```
* {
    stroke: black,lightgray;
    stroke-width: 3,2;
    label: [name];
    font-family: 'Ariel';
    font-size: 10;
    font-fill: black;
    shield: symbol(square);
}
:shield {
    fill: white;
    stroke: black;
    size: 18;
}
```

Antialiasing

Answer for *Explore Antialiasing*:

1. When we rendered our initial preview, without a stroke, thin white gaps (or slivers) are visible between our polygons.



This effect is made more pronounced by the rendering engine making use of the Java 2D sub-pixel accuracy. This technique is primarily used to prevent an aliased (stair-stepped) appearance on diagonal lines.

2. **Explore:** Experiment with **fill** and **stroke** settings to eliminate slivers between polygons.

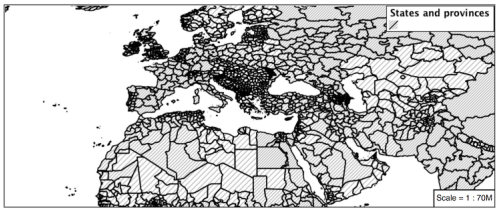
   The obvious approach works - setting both values to the same color:

```
symbolizers:
- polygon:
    stroke-color: 'lightgrey'
    stroke-width: 1
    fill-color: 'lightgrey'
```

Categorize

Answer for *Explore Categorize*:

1. An exciting use of the GeoServer **shape** symbols is the theming by changing the **size** used for pattern density.
2. **Explore:** Use the **Categorize** function to theme by **datarank**.



Example:

```
.. code-block:: css

   * {
     fill: symbol('shape://slash');
     fill-size: [
        Categorize(datarank,
         4, 4,
         5, 6,
         8, 10,
        10)
     ];
     stroke: black;
   }
   :fill {
     stroke: darkgray;
   }
```

Halo

Answer for *Challenge Halo*:

1. The halo example used the fill color and opacity for a muted halo, while this improved readability it did not bring attention to our labels.

   A common design choice for emphasis is to outline the text in a contrasting color.

2. **Challenge:** Produce a map that uses a white halo around black text.

   Here is an example:

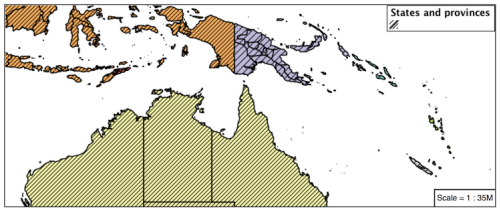
```
* {  stroke: gray;
    fill: #7EB5D3;
    label: [name];
    label-anchor: 0.5 0.5;
    font-fill: black;
    font-family: "Arial";
    font-size: 14;
    halo-radius: 1;
    halo-color: white;
  }
```

Theming using Multiple Attributes

Answer for *Challenge Theming using Multiple Attributes*:

1. A powerful tool is theming using multiple attributes. This is an important concept allowing map readers to perform "integration by eyeball" (detecting correlations between attribute values information).

2. **Challenge:** Combine the **mapcolor9** and **datarank** examples to reproduce the following map.



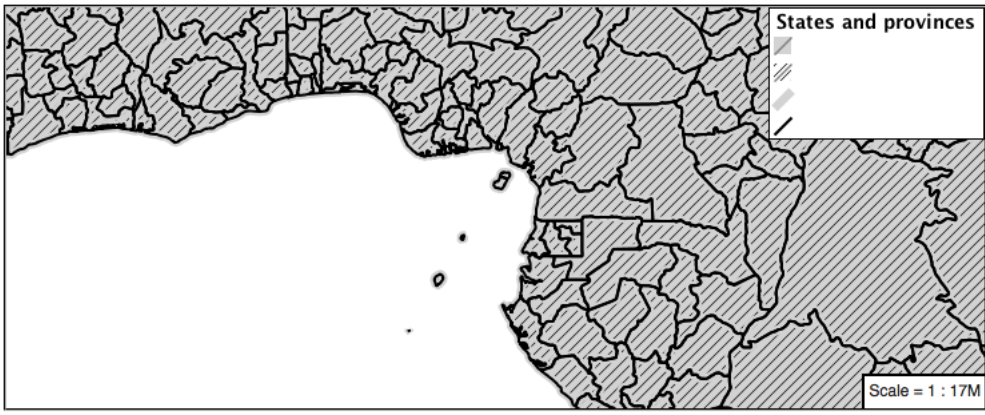
   This should be a cut and paste using the `recode` example, and `categorize` examples already provided.

```
* {
   fill: [
    recode(mapcolor9,
      1,'#8dd3c7', 2,'#ffffb3', 3,'#bebada',
      4,'#fb8072', 5,'#80b1d3', 6,'#fdb462',
      7,'#b3de69', 8,'#fccde5', 9,'#d9d9d9')
   ], symbol('shape://slash');

   fill-size: '',[
      Categorize(datarank,
       6, 4,
       8, 6,
      10, 10,
      12)
   ];
   stroke: black;
}
:fill {
   stroke: black;
}
```

Use of Use of Z-Index

Answer for *Challenge Use of Z-Index*:

1. Earlier we looked at using **z-index** to simulate line string casing. The line work was drawn twice, once with thick line, and then a second time with a thinner line. The resulting effect is similar to text halos - providing breathing space around complex line work allowing it to stand out.
2. **Challenge:** Use what you know of LineString **z-index** to reproduce the following map:



This is a tricky challenge. While it is easy enough to introduce z-index to control stroke what is not immediately obvious is that z-order also controls fill order. The previous examples illustrate how to introduce z-order, some thought is required to untangle fill and stroke z-order (dummy stroke definitions need to be introduced using empty commas).

```
* {
  fill: lightgray, symbol('shape://slash');
  fill-size: 8px;
  stroke: '','',lightgray, black;
  stroke-width: '','',6,1.5;
  z-index: 1,2,3,4;
}
:fill {
  stroke: black;
  stroke-width: 0.75;
}
```

The included legend should be a large clue about what is going on.

Geometry Location

Answer for *Challenge Geometry Location*:

1. The **mark** property can be used to render any geometry content.
2. **Challenge:** Try this yourself by rendering a polygon layer using a **mark** property.

This can be done one of two ways:

- Changing the association of a polygon layer, such as `ne:states_provinces_shp` to point_example and using the layer preview page.
- Changing the *Layer Preview* tab to a polygon layer, such as `ne:states_provinces_shp`.

The important thing to notice is that the centroid of each polygon is used as a point location.

Dynamic Symbolization

Answer for *Explore Dynamic Symbolization*:

1. SLD Mark and ExternalGraphic provide an opportunity for dynamic symbolization.

   This is accomplished by embedding a small CQL expression in the string passed to symbol or url. This sub-expression is isolated with `${ }` as shown:

```
- point:
    symbols:
    - mark:
        shape: ${if_then_else(equalTo(FEATURECLA,
↪'Admin-0 capital'),'star','circle')}
```

2. **Challenge:** Use this approach to rewrite the *Dynamic Styling* example.

   Example available here `point_example.css`

Layer Group

Answer for *Challenge Layer Group*:

1. Use a **Layer Group** to explore how symbology works together to form a map.

- ne:NE1
- ne:states_provincces_shp
- ne:populated_places

2. This background is relatively busy and care must be taken to ensure both symbols and labels are clearly visible.
3. **Challenge:** Do your best to style populated_places over this busy background.

   Here is an example with labels for inspiration:

This should be an opportunity to revisit label halo settings from *Polygons*.

```
* {
   mark-size: [5+((10-SCALERANK)/3)];

   font-fill: black;
   font-family: "Arial";
   font-size: 10;

   label-anchor: 0.5 1;
   label-offset: 0 [-12+SCALERANK];

   halo-radius: 2;
   halo-color: lightgray;
   halo-opacity:0.7;

   mark-label-obstacle: true;
   label-max-displacement: 90;
   label-priority: [0 - LABELRANK];
}
:symbol {
  fill: black;
  stroke: white;
  stroke-opacity:0.75;
}
```

Using a lightgray halo, 0.7 opacity and radius 2 fades out the complexity immediately surrounding the label text improving legibility.

Contrast Enhancement

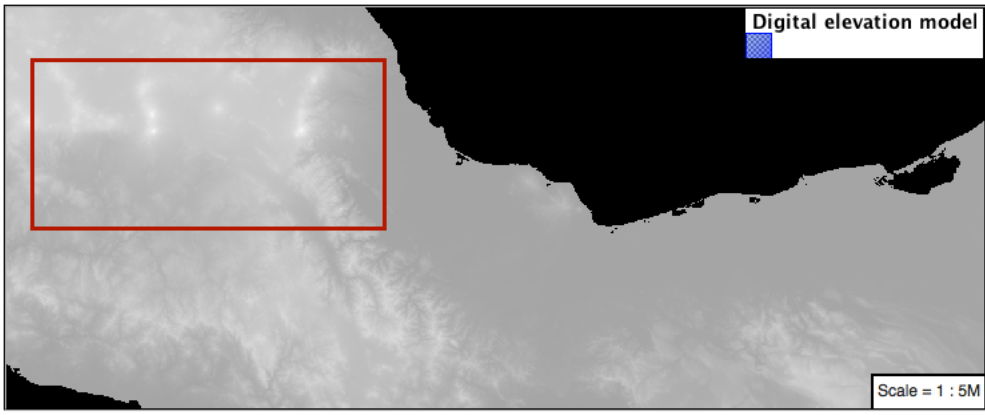
Discussion for *Explore Contrast Enhancement*:

1. A special effect that is effective with grayscale information is automatic contrast adjustment.
2. Make use of a simple contrast enhancement with `usgs:dem`:

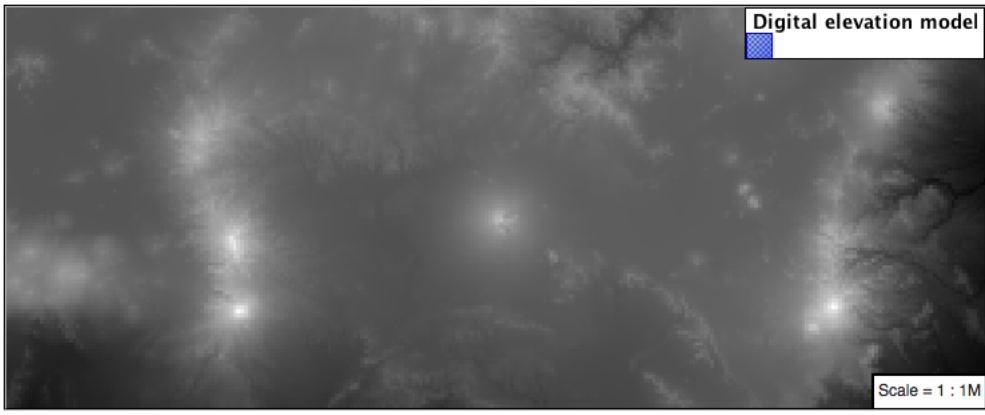
```
* {
    raster-channels: auto;
```

```
                raster-contrast-enhancement: normalize;
            }
```

3. Can you explain what happens when zoom in to only show a land area (as indicated with the bounding box below)?



What happens is insanity, normalize stretches the palette of the output image to use the full dynamic range. As long as we have ocean on the screen (with value 0) the land area was shown with roughly the same presentation.



Once we zoom in to show only a land area, the lowest point on the screen (say 100) becomes the new black, radically altering what is displayed on the screen.

Intervals

Answer for *Challenge Intervals*:

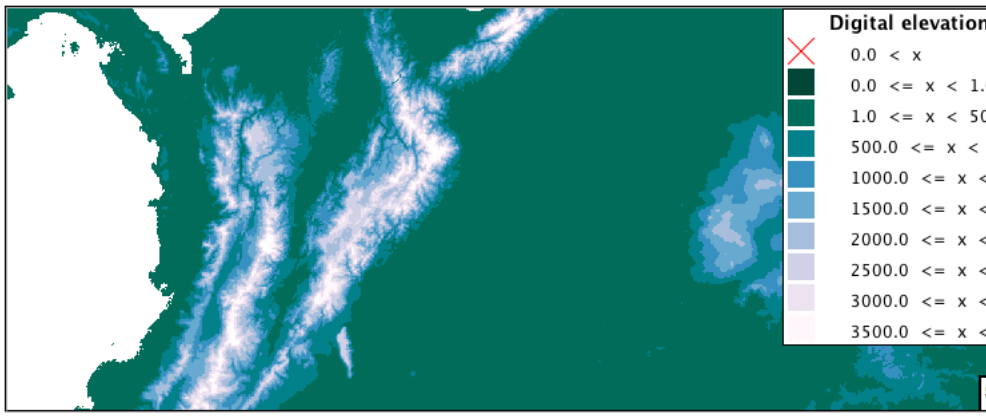
1. The color-map **type** property dictates how the values are used to generate a resulting color.

- `ramp` is used for quantitative data, providing a smooth interpolation between the provided color values.
- `intervals` provides categorization for quantitative data, assigning each range of values a solid color.

- `values` is used for qualitative data, each value is required to have a **color-map** entry or it will not be displayed.

2. **Chalenge:** Update your DEM example to use **intervals** for presentation. What are the advantages of using this approach for elevation data?

   By using intervals it becomes very clear how relatively flat most of the continent is. The ramp presentation provided lots of fascinating detail which distracted from this fact.



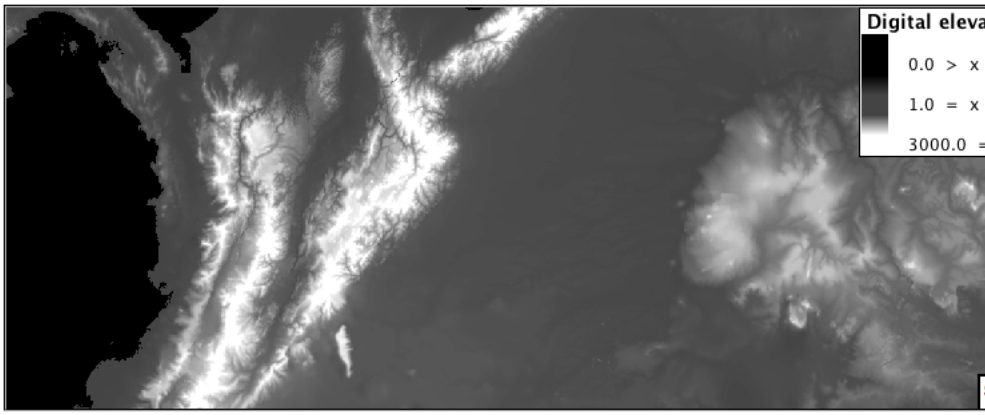
Here is style for you to cut and paste:

```
* {
  raster-channels: auto;
  raster-color-map:
     color-map-entry(#014636,    0,0)
     color-map-entry(#014636,    1)
     color-map-entry(#016c59,  500)
     color-map-entry(#02818a,1000)
     color-map-entry(#3690c0,1500)
     color-map-entry(#67a9cf,2000)
     color-map-entry(#a6bddb,2500)
     color-map-entry(#d0d1e6,3000)
     color-map-entry(#ece2f0,3500)
     color-map-entry(#fff7fb,4000);
  raster-color-map-type: intervals;
}
```

Clear Digital Elevation Model Presentation

Answer for *Challenge Clear Digital Elevation Model Presentation*:

1. Now that you have seen the data on screen and have a better understanding how would you modify our initial gray-scale example?
2. **Challenge:** Use what you have learned to present the `usgs:dem` clearly.



The original was a dark mess. Consider making use of mid-tones (or adopting a sequential palette from color brewer) in order to fix this. In the following example the ocean has been left dark, allowing the mountains stand out more.

```
* {
  raster-channels: auto;
  raster-color-map: color-map-entry(#000000, 0)
                    color-map-entry(#444444, 1)
                    color-map-entry(#FFFFFF, 3000);
}
```

Raster Opacity

Discussion for *Challenge Clear Digital Elevation Model Presentation*:

1. There is a quick way to make raster data transparent, raster **opacity** property works in the same fashion as with vector data. The raster as a whole will be drawn partially transparent allow content from other layers to provide context.
2. **Challenge:** Can you think of an example where this would be useful?

   This is difficult as raster data is usually provided for use as a basemap, with layers being drawn over top.

   The most obvious example here is the display of weather systems, or model output such as fire danger. By drawing the raster with some transparency, the landmass can be shown for context.

6.7.4 YSLD Styling Workbook

GeoServer styling can be used for a range of creative effects. This section introduces the *YSLD Extension* which can be used as alternative to SLD files.

YSLD Quickstart

In the last section, we saw how the OGC defines style using XML documents (called SLD files).

We will now explore GeoServer styling in greater detail using a tool to generate our SLD files. The **YSLD** GeoServer extension is used to generate SLD files using a clearer, more consice language based on **YAML**. Unlike **CSS**, the **YSLD** styling language has a one-to-one correspondance to SLD, meaning that each line of YSLD translates directly to one or more lines of SLD. Aditionally, A YSLD style can be converted to SLD *and back* without loss of information.

Using the YSLD extension to define styles results in shorter examples that are easier to understand. At any point we will be able to review the generated SLD file.

Reference:

- *YSLD Reference*

Syntax

This section provides a quick introduction to YSLD syntax for mapping professionals who may not be familiar with YAML.

Property Syntax

Individual statements (or directives) in a YSLD styling document are designed as key-value, or property-value pairs of the following form:

```
<property>: <value>
```

The `<property>` is a string denoting the property name, while the `<value>` can be one of a number of different types depending on context.

Integer	Numerical value. May be surrounded by quotes.
Float	Numerical value. May be surrounded by quotes.
Text	Text value. If value is amiguous, use single quotes.
Color	Hexadecimal color of the form `'#RRGGBB'`.
Tuple	A list of values in brackets. e.g. `[0, 1]`
Expression	CQL expression surrounded by `${ }`

Mappings and lists

There are three types of objects in a YSLD document:

1. Scalar, a simple value
2. Mapping, a collection of key-value (property-value) pairs

3. List, any collection of objects. A list can contain mappings, scalars, and even other lists.

   Lists require dashes for every entry, while mappings do not.

   For example, a symbolizer block is a list, so every entry requires its own dash:

   The `polygon:` and `text:` objects (the individual symbolizers themselves) are mappings, and as such, the contents do not require dashes, only indents:

```
- polygon:
    stroke-color: '#808080'
    fill-color: '#FF0000'
```

   The dash next to polygon means that the item itself is contained in a list, not that it contains a list. And the placement of the dash is at the same level of indentation as the list title.

   If you have a list that contains only one item, and there is no other content at higher levels of the list, you may omit the enclosing elements. For example, the following are equivalent:

```
feature-styles:
- rules:
  - symbolizers:
    - point:
        symbols:
        - mark:
            shape: circle
            fill-color: 'gray'
```

```
point:
  symbols:
  - mark:
      shape: circle
      fill-color: 'gray'
```

   This is usefull for making your styles more concise.

Indentation

Indentation is very important in YSLD. All directives must be indented to its proper place to ensure proper hierarchy. Improper indentation will cause a style to be rendered incorrectly, or not at all.

For example, the polygon symbolizer, since it is a mapping, contains certain parameters inside it, such as the color of the fill and stroke. These must be indented such that they are "inside" the polygon block.

In this example, the following markup is **correct**:

```
- polygon:
    fill-color: '#808080'
```

```
      fill-opacity: 0.5
      stroke-color: black
      stroke-opacity: 0.75
```

The parameters inside the polygon (symbolizer) are indented, meaning that they are referencing the symbolizer and are not "outside it."

Compare to the following **incorrect** markup:

```
- polygon:
  fill-color: '#808080'
  fill-opacity: 0.5
  stroke-color: black
  stroke-opacity: 0.75
```

Rules

We have already seen a CSS style composed of a single rule:

```
point:
  symbols:
  - mark:
      shape: circle
      fill-color: 'gray'
```

We can make a style consisting of more than one rule, carefully choosing the selector for each rule. In this case we are using a selector to style capital cities with a star, and non-capital with a circle:

```
rules:
  - filter: ${FEATURECLA = 'Admin-0 capital'}
    scale: [min, max]
    symbolizers:
    - point:
        size: 6
        symbols:
        - mark:
            shape: star
            stroke-color: 'black'
            stroke-width: 1
            fill-color: 'gray'
  - filter: ${FEATURECLA <> 'Admin-0 capital'}
    scale: [min, max]
    symbolizers:
    - point:
        size: 6
        symbols:
        - mark:
            shape: circle
            stroke-color: 'black'
            stroke-width: 1
            fill-color: 'gray'
```

The feature attribute test performed above uses **Constraint Query Language (CQL)**. This syntax can be used to define filters to select

content, similar to how the SQL WHERE statement is used. It can also be used to define expressions to access attribute values allowing their use when defining style properties.

Rule selectors can also be triggered based on the state of the rendering engine. In this example we are only applying labels when zoomed in:

```
rules:
  - scale: [min, '2.0E7']
    symbolizers:
    - text:
        label: ${NAME}
        fill-color: 'gray'
```

In the above example the label is defined using the CQL Expression `NAME`. This results in a dynamic style that generates each label on a case-by-case basis, filling in the label with the feature attribute `NAME`.

Reference:

- Filter Syntax (YSLD Reference)
- *ECQL Reference* (User Guide)

Variables

Up to this point we have been styling individual features, documenting how each shape is represented.

When styling multiple feaures, or using filters to style individual features in different yars, you may need to repeat styling information.

Variables in YSLD allow for a certain directive or block of directives to be defined by name and later reused. This can greatly simplify the styling document.

The two most-common use cases for using variables are:

- To create a more-friendly name for a value (such as using myorange instead of `#EE8000`)
- To define a block of directives to remove redundant content and to decrease file length

It is customary, but not required, to place all definitions at the very top of the YSLD file.

The syntax for defining a variable as a single value is:

```
define: &variable <value>
```

The defined variable can then be used as a value by variable name with a `*`:

```
<directive>: *variable
```

The syntax for defining a variable as a content block is:

```
define: &varblock
  <directive>: <value>
  <directive>: <value>
  ...
  <block>:
  - <directive>: <value>
    <directive>: <value>
  ...
```

The syntax for using a variable block is to prepend the variable name with `<<: *`. For example:

```
<block>:
- <directive>: <value>
  <<: *varblock
```

- Variables (YSLD Reference)

Compare YSLD to SLD

As noted above, YSLD has a one-to-one correspondance with SLD, it merely uses a different markup language to diplay the same content. We can compare a SLD style with a YSLD style to see this correspondence:

SLD Style

Here is an example `SLD file` for reference:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<StyledLayerDescriptor version="1.0.0"
 xsi:schemaLocation="http:/
↪/www.opengis.net/sld StyledLayerDescriptor.xsd"
 xmlns="http://www.opengis.net/sld"
 xmlns:ogc="http://www.opengis.net/ogc"
 xmlns:xlink="http://www.w3.org/1999/xlink"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <NamedLayer>
    <Name>airports</Name>
    <UserStyle>
      <Title>Airports</Title>
      <FeatureTypeStyle>
        <Rule>
          <Name>airports</Name>
          <Title>Airports</Title>
          <PointSymbolizer>
            <Graphic>
              <ExternalGraphic>
                <OnlineResource xlink:type="simple"
                xlink:href="airport.svg" />
                <Format>image/svg</Format>
              </ExternalGraphic>
              <Size>16</Size>
            </Graphic>
          </PointSymbolizer>
```

```
26                    </Rule>
27                  </FeatureTypeStyle>
28                </UserStyle>
29              </NamedLayer>
30            </StyledLayerDescriptor>
```

YSLD Style

Here is the same example as `YSLD`:

```
1   name: airports
2   title: Airports
3   scale: [min, max]
4   symbolizers:
5   - point:
6       size: 16
7       symbols:
8       - external:
9           url: airport.svg
10          format: image/svg
```

We use a point symbolizer to indicate we want this content drawn as a **Point** (line 16 in the SLD, line 5 in the YSLD). The point symbolizer declares an external graphic, which contains the URL `airports.svg` indicating the image that should be drawn (line 20 in the SLD, line 9 in the YSLD).

Tour

To confirm everything works, let's reproduce the airports style above.

1. Navigate to the **Styles** page.
2. Each time we edit a style, the contents of the associated SLD file are replaced. Rather then disrupt any of our existing styles we will create a new style. Click *Add a new style* and choose the following:

Name:	`airports0`
Workspace:	(leave empty)
Format:	`YSLD`

3. Replace the initial YSLD definition with with our airport YSLD example and click *Apply*:

```
name: airports
title: Airports
scale: [min, max]
symbolizers:
- point:
    size: 16
    symbols:
    - external:
```

```
                                    url: airport.svg
                                    format: image/svg
```

4. Click the *Layer Preview* tab to preview the style. We want to preview on the aiports layer, so click the name of the current layer and select `ne:airports` from the list that appears. You can use the mouse buttons to pan and scroll wheel to change scale.

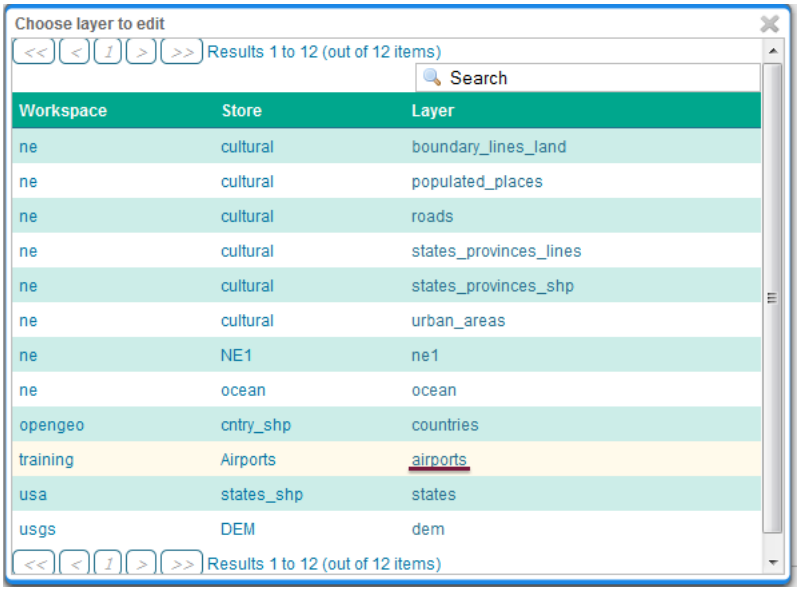


Fig. 6.300: Choosing the airports layer

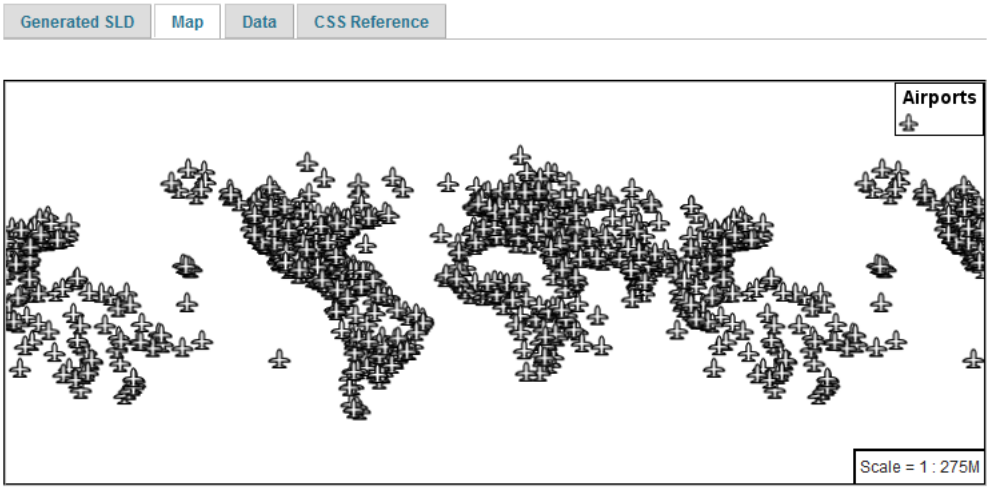
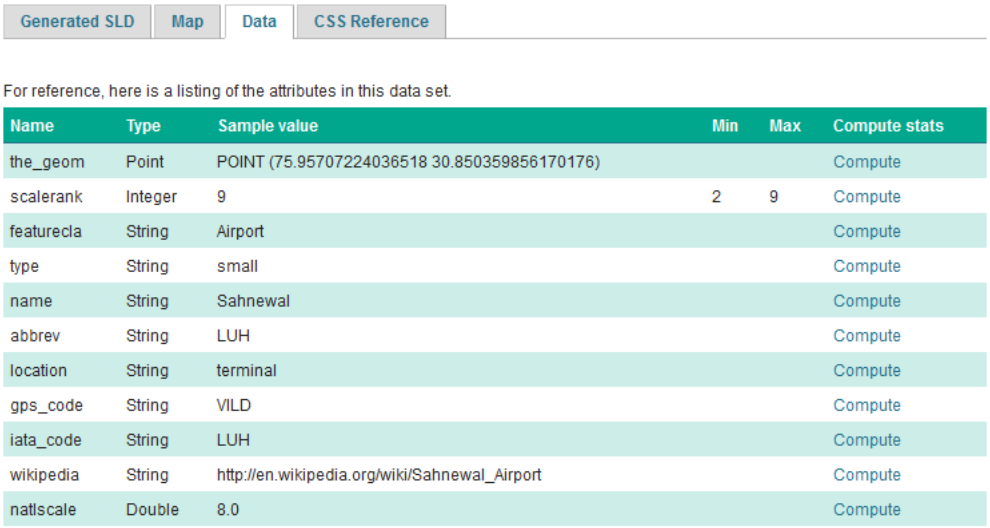


Fig. 6.301: Layer preview

5. Click *Layer Data* for a summary of the selected data.



Name	Type	Sample value	Min	Max	Compute stats
the_geom	Point	POINT (75.95707224036518 30.850359856170176)			Compute
scalerank	Integer	9	2	9	Compute
featurecla	String	Airport			Compute
type	String	small			Compute
name	String	Sahnewal			Compute
abbrev	String	LUH			Compute
location	String	terminal			Compute
gps_code	String	VILD			Compute
iata_code	String	LUH			Compute
wikipedia	String	http://en.wikipedia.org/wiki/Sahnewal_Airport			Compute
natlscale	Double	8.0			Compute

Fig. 6.302: Layer attributes

Bonus

Finished early? For now please help your neighbour so we can proceed with the workshop.

If you are really stuck please consider the following challenge rather than skipping ahead.

Explore Data

1. Return to the *Data* tab and use the *Compute* link to determine the minimum and maximum for the **scalerank** attribute.

Challenge Compare SLD Generation

The rest API can be used to review your YAML file directly.

Browser:

- view-source:http://localhost:8080/geoserver/rest/styles/airport0.yaml

Command line:

```
curl -v -u admin:geoserver -XGET http:/
↪/localhost:8080/geoserver/rest/styles/airports0.yaml
```

1. The REST API can also be used generate an SLD file:

Browser:

- view-source:http://localhost:8080/geoserver/rest/styles/airport0.sld?pretty=true

Command line:

```
curl -v -u admin:geoserver -XGET http://localhost:8080/
↪geoserver/rest/styles/airports0.sld?pretty=true
```

1. Compare the generated SLD differ above with the hand written `SLD file` used as an example?

   Challenge: What differences can you spot?

Lines

We will start our tour of YSLD styling by looking at the representation of lines.

Fig. 6.303: LineString Geometry

Review of line symbology:

- Lines can be used to represent either abstract concepts with length but not width such as networks and boundaries, or long thin features with a width that is too small to represent on the map. This means that **the visual width of line symbols do not normally change depending on scale.**
- Lines are recorded as LineStrings or Curves depending on the geometry model used.
- SLD uses a **LineSymbolizer** to record how the shape of a line is drawn. The primary characteristic documented is the **Stroke** used to draw each segment between vertices.
- Labeling of line work is anchored to the mid-point of the line. GeoServer provides a vendor option to allow label rotation aligned with line segments.

  For our exercises we are going to be using simple YSLD documents, often consisting of a single rule, in order to focus on the properties used for line symbology.

  Each exercise makes use of the `ne:roads` layer.

  Reference:

- *YSLD Reference*
- *YSLD Reference Line symbolizer* (User Manual | YSLD Reference)
- *LineString* (User Manual | SLD Reference)

Line

A line symbolizer is represented by a `line` key. You can make a completely default symbolizer by giving it an empty map

```
line:
```

1. Navigate to the **Styles** page.

Fig. 6.304: Basic Stroke Properties

2. Click *Add a new style* and choose the following:

New style name:	`line_example`
Workspace for new layer:	Leave blank
Format:	`YSLD`

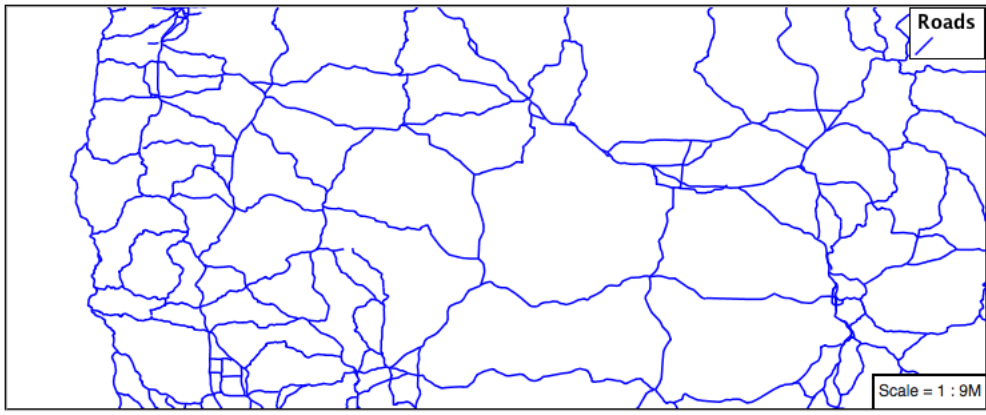
3. Choose *line* from the `Generate a default style` dropdown and click *generate*.
4. The style editor should look like below:

```
title: dark yellow line
symbolizers:
- line:
    stroke-width: 1.0
    stroke-color: '#99cc00'
```

Note: The title and value for **stroke-color** may be different.

1. Click *Apply*
2. Click *Layer Preview* to see your new style applied to a layer.

   You can use this tab to follow along as the style is edited, it will refresh each time *Apply* is pressed.



3. You can see the equivalent SLD by requesting `http://localhost:8080/geoserver/rest/styles/line_example.sld?pretty=true` which will currently show the default line symbolizer we created.

```
<?xml version="1.0" encoding="UTF-8"?>
<sld:StyledLayerDescriptor xmlns="http://www.opengis.
net/sld" xmlns:sld="http://www.opengis.net/sld
" xmlns:gml="http://www.opengis.net/gml" xmlns:ogc=
"http://www.opengis.net/ogc" version="1.0.0">
  <sld:NamedLayer>
    <sld:Name>line_example</sld:Name>
```

```
      <sld:UserStyle>
        <sld:Name>line_example</sld:Name>
        <sld:Title>dark yellow line</sld:Title>
        <sld:FeatureTypeStyle>
          <sld:Name>name</sld:Name>
          <sld:Rule>
            <sld:LineSymbolizer>
              <sld:Stroke>
                <sld:CssParameter␣
↪name="stroke">#99CC00</sld:CssParameter>
              </sld:Stroke>
            </sld:LineSymbolizer>
          </sld:Rule>
        </sld:FeatureTypeStyle>
      </sld:UserStyle>
    </sld:NamedLayer>
  </sld:StyledLayerDescriptor>
```

We only specified the line symbolizer, so all of the boilerplate around was generated for us.

1. Additional properties can be used fine-tune appearance. Use **stroke-color** to specify the colour of the line.

```
line:
  stroke-color: blue
```

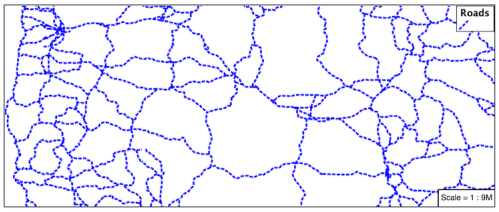
2. **stroke-width** lets us make the line wider

```
line:
  stroke-color: blue
  stroke-width: 2px
```

3. **stroke-dasharray** applies a dot dash pattern.

```
line:
  stroke-color: blue
  stroke-width: 2px
  stroke-dasharray: 5 2
```

4. Check the *Layer Preview* tab to preview the result.



Note: The GeoServer rendering engine is quite sophisticated and allows the use of units of measure (such as `m` or `ft`). While we are using pixels in this example, real world units will be converted using the current scale, allowing for lines that change width with the scale.

Multiple Symbolizers

Providing two strokes is often used to provide a contrasting edge (called casing) to thick lines. This can be created using two symbolizers.

1. Start by filling in a bit of boilerplate that we'll be using

```
feature-styles:
- rules:
  - symbolizers:
    - line:
        stroke-color: '#8080E6'
        stroke-width: 3px
```

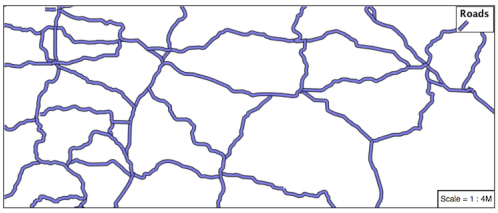
The line symbolizer is inside a rule, which is inside a feature style.

2. Add a second symbolizer to the rule

```
feature-styles:
- rules:
  - symbolizers:
    - line:
        stroke-color: black
        stroke-width: 5px
    - line:
        stroke-color: '#8080E6'
        stroke-width: 3px
```

The wider black line is first so it is drawn first, then the thinner blue line drawn second and so over top of the black line. This is called the painter's algorithm.

3. If you look carefully you can see a problem with our initial attempt. The junctions of each line show that the casing outlines each line individually, making the lines appear randomly overlapped. Ideally we would like to control this process, only making use of this effect for overpasses.



This is becaue the black and blue symbolizers are being drawn on a feature by feature basis. For nice line casing, we want all of the black symbols, and then all of the blue symbols.

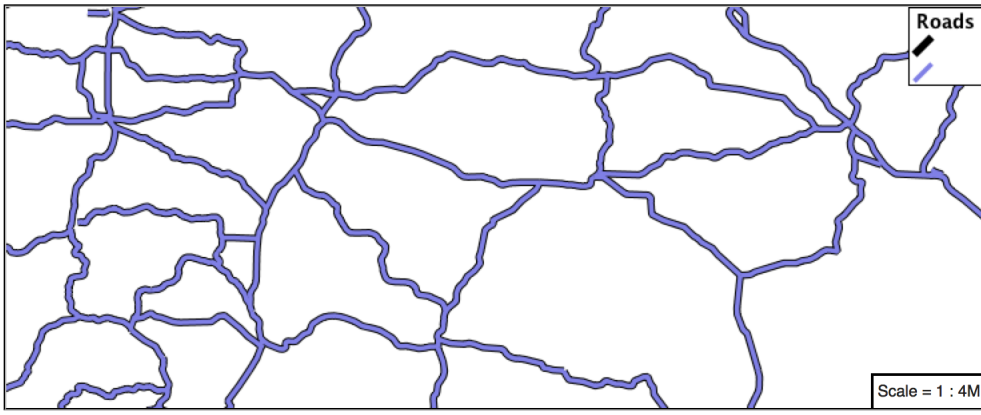
4. Create a new feature style and move the second symbolizer there.

```
feature-styles:
- rules:
  - symbolizers:
    - line:
```

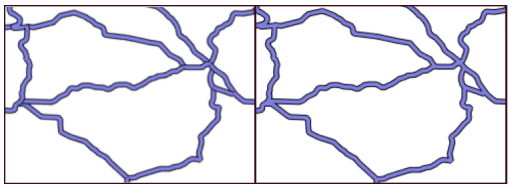
```
                stroke-color: black
                stroke-width: 5px
    - rules:
      - symbolizers:
        - line:
            stroke-color: '#8080E6'
            stroke-width: 3px
```

Again we are using painter's algorithm order: the first feature style is drawn first then the second so the the second is drawn on top of the first. The difference is that for each feature style, all of the features are drawn before the next feature style is drawn.

5. If you look carefully you can see the difference.



6. By using **feature styles** we have been able to simulate line casing.



Label

Our next example is significant as it introduces how text labels are generated.

Fig. 6.305: Use of Label Property

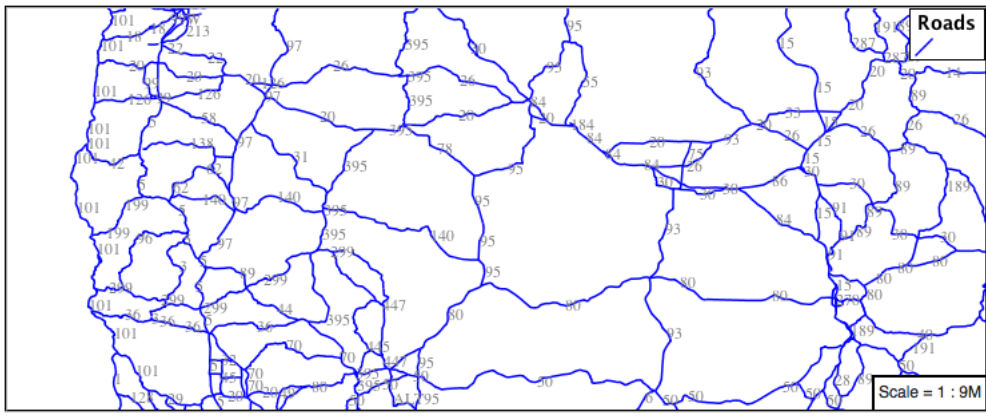
This is also our first example making use of a dynamic style (where a value comes from an attribute from your data).

1. To enable LineString labeling we add a `text` symbolizer witrh a `label`.

   Update `line_example` with the following:

```
symbolizers:
- line:
    stroke-color: blue
    stroke-width: 1px
- text:
    label: ${name}
```

2. The SLD standard documents the default label position for each kind of Geometry. For LineStrings the initial label is positioned on the midway point of the line.



3. We have used an expression to calculate a property value for label. The **label** is generated dynamically from the `name` attribute. Expressions are supplied within curly braces preceded with a dollar sign, and use Extended Constraint Query Language (ECQL) syntax.

```
symbolizers:
- line:
    stroke-color: blue
    stroke-width: 1px
- text:
    label: ${name}
```

4. Additional keys can be supplied to fine-tune label presentation:

```
symbolizers:
- line:
    stroke-color: blue
    stroke-width: 1px
- text:
    label: ${name}
    fill-color: black
    placement: line
    offset: 7px
```

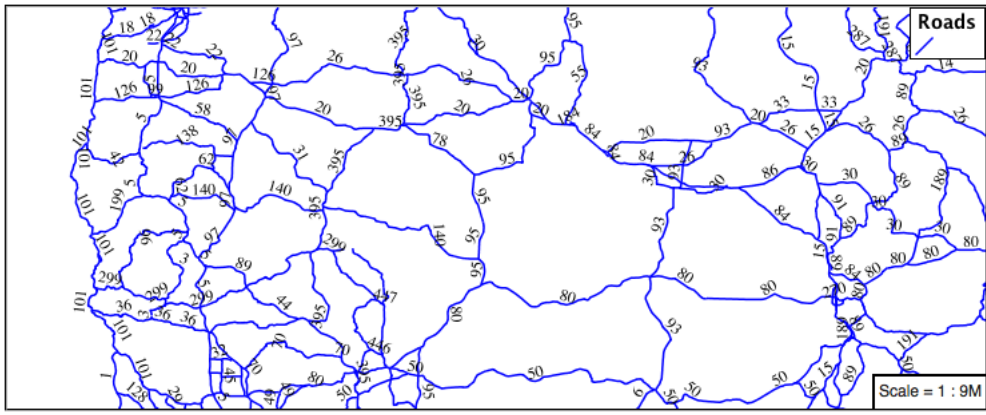
5. The **fill-color** property is set to `black` to provide the colour of the text.

```
symbolizers:
- line:
    stroke-color: blue
    stroke-width: 1px
- text:
```

```
    label: ${name}
    fill-color: black
    placement: line
    offset: 7px
```

6. The **placement** property is used to set how the label is placed with respect to the line. By default it is `point` which casues the label to be placed next to the midpoint as it would be for a point feature. When set to `line` it is placed along the line instead. **offset** specifies how far from the line the label should be placed.

```
symbolizers:
- line:
    stroke-color: blue
    stroke-width: 1px
- text:
    label: ${name}
    fill-color: black
    placement: line
    offset: 7px
```



7. When using point placement, you can shift the position of the label using **displacement** instead of **offset**. This takes an x value and a y value.

```
symbolizers:
- line:
    stroke-color: blue
    stroke-width: 1px
- text:
    label: ${name}
    fill-color: black
    displacement: [5px, -10px]
```

How Labeling Works

The rendering engine collects all the generated labels during the rendering of each layer. Then, during labeling, the engine sorts through the labels performing collision avoidance (to prevent labels overlapping). Finally the rendering engine draws the labels on top of the

map. Even with collision avoidance you can spot areas where labels are so closely spaced that the result is hard to read.

The parameters provided by SLD are general purpose and should be compatible with any rendering engine.

To take greater control over the GeoServer rendering engine we can use "vendor specific" parameters. These hints are used specifically for the GeoServer rendering engine and will be ignored by other systems. In YSLD vendor specific parameters start with the prefix **x-**.

1. The ability to take control of the labeling process is exactly the kind of hint a vendor specific parameter is intended for.

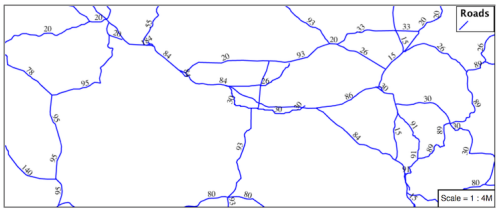
   Update `line_example` with the following:

```
symbolizers:
- line:
    stroke-color: blue
    stroke-width: 1px
- text:
    label: ${name}
    fill-color: black
    placement: line
    offset: 7px
    x-label-padding: 10
```

2. The parameter **x-label-padding** provides additional space around our label for use in collision avoidance.

```
symbolizers:
- line:
    stroke-color: blue
    stroke-width: 1px
- text:
    label: ${name}
    fill-color: black
    placement: line
    offset: 7px
    x-label-padding: 10
```

3. Each label is now separated from its neighbor, improving legibility.



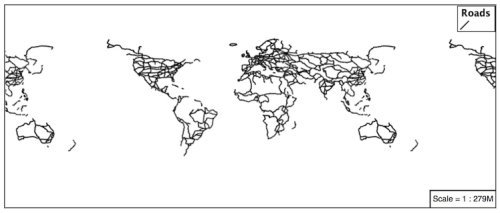
Scale

This section explores the use of rules with filters and scale restrictions.

1. Replace the *line_example* YSLD definition with:

```
rules:
- filter: ${scalerank < 4}
  symbolizers:
  - line:
      stroke-color: black
      stroke-width: 1
```

2. And use the *Layer Preview* tab to preview the result.



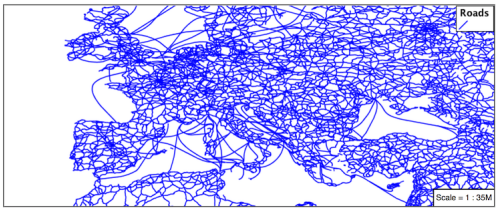
3. The **scalerank** attribute is provided by the Natural Earth dataset to allow control of the level of detail based on scale. Our filter short-listed all content with scalerank 4 or lower, providing a nice quick preview when we are zoomed out.

4. In addition to testing feature attributes, selectors can also be used to check the state of the rendering engine.

   Replace your YSLD with the following:

```
rules:
- scale: [35000000, max]
  symbolizers:
  - line:
      stroke-color: black
      stroke-width: 1
- scale: [min, 35000000]
  symbolizers:
  - line:
      stroke-color: blue
      stroke-width: 1
```

5. As you adjust the scale in the *Layer Preview* (using the mouse scroll wheel) the color will change between black and blue. You can read the current scale in the bottom right corner, and the legend will change to reflect the current style.



6. Putting these two ideas together allows control of level detail based on scale:

```
define: &primaryStyle
  stroke-color: black
define: &primaryFilter ${scalerank <= 4}
```

```
define: &secondaryStyle
  stroke-color: '#000055'
define: &secondaryFilter ${scalerank = 5}

rules:

  - else: true
    scale: [min, 9000000]
    symbolizers:
    - line:
        stroke-color: '#888888'
        stroke-width: 1

  - filter: ${scalerank = 7}
    scale: [min, 17000000]
    symbolizers:
    - line:
        stroke-color: '#777777'
        stroke-width: 1

  - filter: ${scalerank = 6}
    scale: [min, 35000000]
    symbolizers:
    - line:
        stroke-color: '#444444'
        stroke-width: 1

  - filter: *secondaryFilter
    scale: [9000000, 70000000]
    symbolizers:
    - line:
        <<: *secondaryStyle
        stroke-width: 1
  - filter: *secondaryFilter
    scale: [min, 9000000]
    symbolizers:
    - line:
        <<: *secondaryStyle
        stroke-width: 2

  - filter: *primaryFilter
    scale: [35000000, max]
    symbolizers:
    - line:
        <<: *primaryStyle
        stroke-width: 1
  - filter: *primaryFilter
    scale: [9000000, 35000000]
    symbolizers:
    - line:
        <<: *primaryStyle
        stroke-width: 2
  - filter: *primaryFilter
    scale: [min, 9000000]
    symbolizers:
    - line:
        <<: *primaryStyle
        stroke-width: 4
```

7. When a rule has both a filter and a scale, it will trigger when both are true.

   The first rule has *else: true* instead of a filter. This causes it to be applied after all other rules have been checked if none of them worked.

   Since there are some things we need to specify more than once like the colour and filter for primary and secondary roads, even as they change size at different scales, they are given names with *define* so they can be reused. The filters are inserted inline using **name* while the style is inserted as a block with *<<: *name*

Bonus

Finished early? Here are some opportunities to explore what we have learned, and extra challenges requiring creativity and research.

In a classroom setting please divide the challenges between teams (this allows us to work through all the material in the time available).

Explore Vendor Option Follow Line

Vendor options can be used to enable some quite spectacular effects, while still providing a style that can be used by other applications.

1. Update *line_example* with the following:

```
symbolizers:
- line:
    stroke-color: '#EDEDFF'
    stroke-width: 10
- text:
    label: '${level} #${name}'
    fill-color: '#000000'
    x-followLine: true
```

   The # character is the comment character in YAML, so we have to quote strings that contain it like colours and in this expression.

2. The property **stroke-width** has been used to make our line thicker in order to provide a backdrop for our label.

```
symbolizers:
- line:
    stroke-color: '#EDEDFF'
    stroke-width: 10
- text:
```

```
      label: '${level} #${name}'
      fill-color: '#000000'
      placement: point
      x-followLine: true
```

3. The **label** property combine several CQL expressions together for a longer label.

```
symbolizers:
- line:
    stroke-color: '#EDEDFF'
    stroke-width: 10
- text:
    label: '${level} #${name}'
    fill-color: '#000000'
    x-followLine: true
```

The expressions in the **label** property:

```
${level} #${name}
```

are inserted into the text by combining them with the text between them using **Concatenate** function:

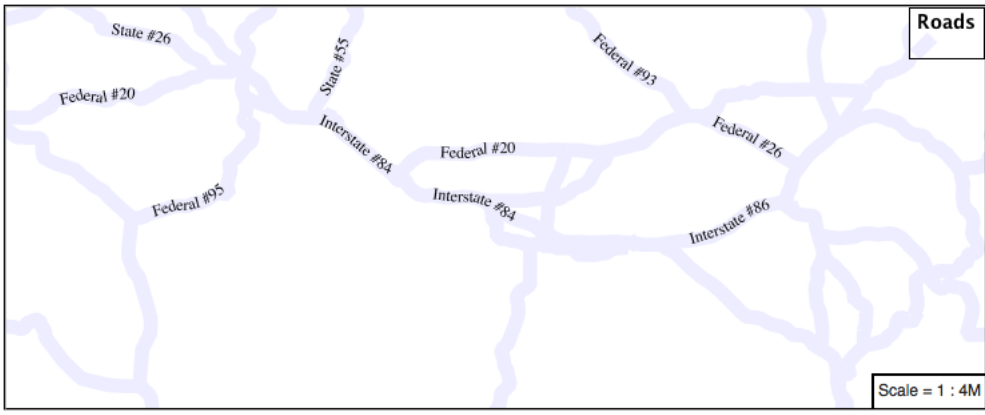
```
[Concatenate(level,' #', name)]
```

This happens silently in the background.

4. The property **x-followLine** provides the ability of have a label exactly follow a LineString character by character.

```
symbolizers:
- line:
    stroke-color: '#EDEDFF'
    stroke-width: 10
- text:
    label: ${level}  ${name}
    fill-color: '#000000'
    x-followLine: true
```

5. The result is a new appearance for our roads.

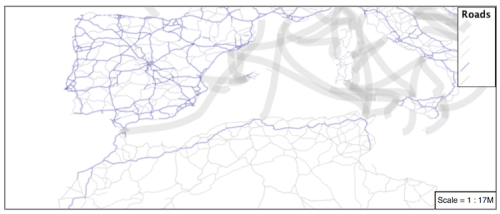


Challenge Classification

1. The roads **type** attribute provides classification information.

   You can **Layer Preview** to inspect features to determine available values for type.

2. **Challenge:** Create a new style adjust road appearance based on **type**.



   note:: The available values are 'Major Highway','Secondary Highway','Road' and 'Unknown'.

   note:: Answer *provided* at the end of the workbook.

Challenge One Rule Classification

1. You can save a lot of typing by doing your classification in an expression using arithmetic or the `Recode` function

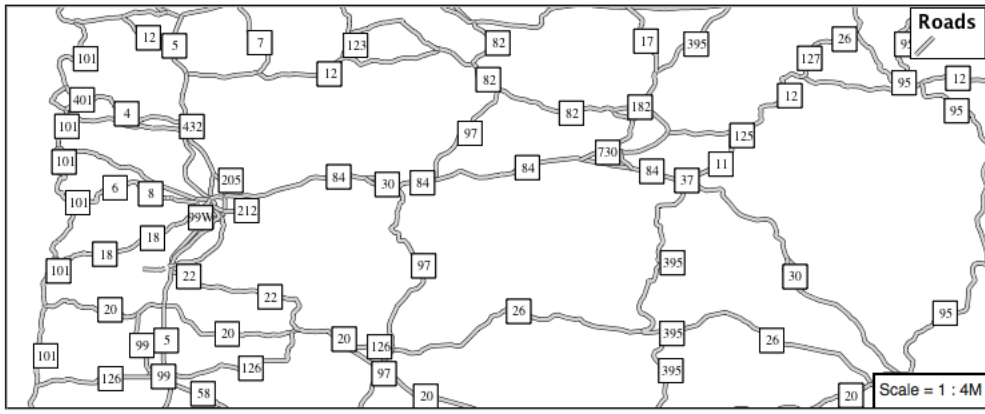
2. **Challenge:** Create a new style and classify the roads based on their scale rank using expressions in a single rule instead of multiple rules with filters.

Note: Answer *provided* at the end of the workbook.

Challenge Label Shields

1. The traditional presentation of roads in the US is the use of a shield symbol, with the road number marked on top.

2. *Challenge:* Have a look at the documentation for putting a graphic on a text symbolizer in SLD and reproduce this technique in YSLD.



Note: Answer *provided* at the end of the workbook.

Polygons

Next we look at how YSLD styling can be used to represent polygons.

Fig. 6.306: Polygon Geometry

Review of polygon symbology:

- Polygons offer a direct representation of physical extent or the output of analysis.
- The visual appearance of polygons reflects the current scale.
- Polygons are recorded as a LinearRing describing the polygon boundary. Further LinearRings can be used to describe any holes in the polygon if present.

  The Simple Feature for SQL Geometry model (used by GeoJSON) represents these areas as Polygons, the ISO 19107 geometry model (used by GML3) represents these areas as Surfaces.
- SLD uses a **PolygonSymbolizer** to describe how the shape of a polygon is drawn. The primary characteristic documented is the **Fill** used to shade the polygon interior. The use of a **Stroke** to describe the polygon boundary is optional.
- Labeling of a polygon is anchored to the centroid of the polygon. GeoServer provides a vendor-option to allow labels to line wrap to remain within the polygon boundaries.

  For our Polygon exercises we will try and limit our YSLD documents to a single rule, in order to showcase the properties used for rendering.

  Reference:

- *YSLD Reference*

- *YSLD Reference Polygon symbolizer* (User Manual | YSLD Reference)
- *Polygons* (User Manual | SLD Reference)

This exercise makes use of the `ne:states_provinces_shp` layer.

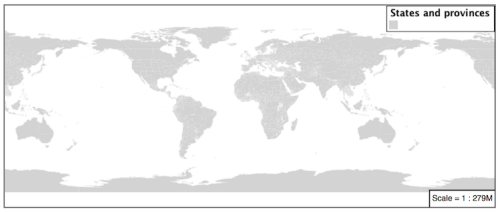
1. Navigate to *Styles*.
2. Create a new style `polygon_example`.

Name:	`polygon_example`
Workspace:	`No workspace`
Format:	`YSLD`

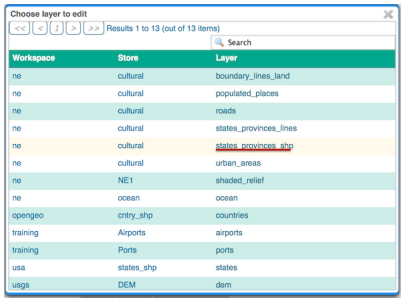
3. Choose *polygon* from the `Generate a default style` drop-down and click *generate*.
4. Replace the generated style with the following style and click *Apply* to save:

```
symbolizers:
- polygon:
    fill-color: 'lightgrey'
```

5. Click on the tab *Layer Preview* to preview.



6. Set `ne:states_provinces_shp` as the preview layer.



Stroke and Fill

The **polygon** symbolizer controls the display of polygon data.

The **fill-color** property is used to provide the color used to draw the interior of a polygon.

1. Replace the contents of `polygon_example` with the following **fill** example:

```
symbolizers:
- polygon:
    fill-color: 'gray'
```

2. The *Layer Preview* tab can be used preview the change:

3. To draw the boundary of the polygon the **stroke** property is used:

   The **stroke** property is used to provide the color and size of the polygon boundary. It is effected by the same parameters (and vendor specific parameters) as used for LineStrings.

```
symbolizers:
- polygon:
    fill-color: 'gray'
    stroke-color: 'black'
    stroke-width: 2
```

Note: Technically the boundary of a polygon is a specific case of a LineString where the first and last vertex are the same, forming a closed LinearRing.

4. The effect of adding **stroke** is shown in the map preview:

5. An interesting technique when styling polygons in conjunction with background information is to control the fill opacity.

   The **fill-opacity** property is used to adjust transparency (provided as range from 0.0 to 1.0). Use of **fill-opacity** to render polygons works well in conjunction with a raster base map. This approach allows details of the base map to shown through.

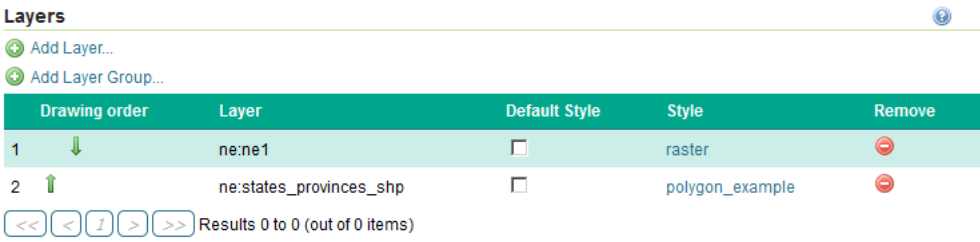
   The **stroke-opacity** property is used in a similar fashion, as a range from 0.0 to 1.0.

```
symbolizers:
- polygon:
    fill-color: 'white'
    fill-opacity: 0.5
    stroke-color: 'lightgrey'
```

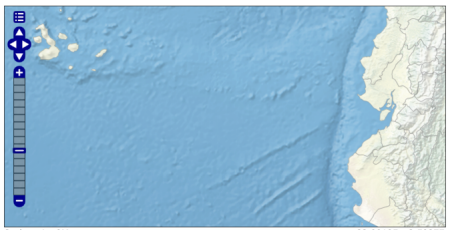
```
                stroke-width: 0.25
                stroke-opacity: 0.5
```

6. As shown in the map preview:

7. This effect can be better appreciated using a layer group.



Where the transparent polygons is used lighten the landscape provided by the base map.



Pattern

The **fill-graphic** property can be used to provide a pattern.

The fill pattern is defined by repeating one of the built-in symbols, or making use of an external image.

1. We have two options for configuring a **fill-graphic** with a repeating graphic:

   Using **external** to reference to an external graphic.

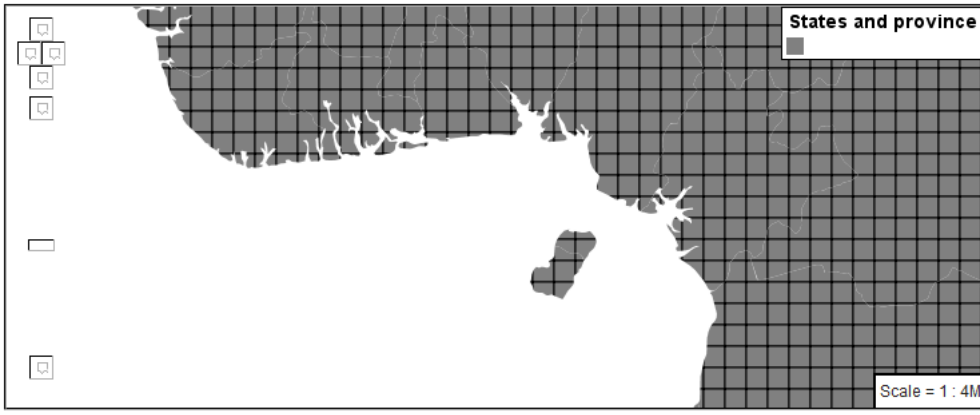
   Use of **mark** to access a predefined shape. SLD provides several well-known shapes (circle, square, triangle, arrow, cross, star, and

x). GeoServer provides additional shapes specifically for use as fill patterns.

Update *polygon_example* with the following built-in symbol as a repeating fill pattern:

```
symbolizers:
- polygon:
    fill-graphic:
      symbols:
      - mark:
          shape: square
          fill-color: 'gray'
          stroke-color: 'black'
          stroke-width: 1
```

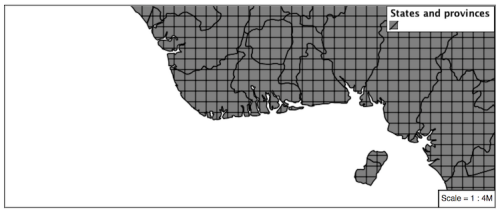
2. The map preview (and legend) will show the result:



3. Add a black stroke:

```
symbolizers:
- polygon:
    stroke-color: 'black'
    stroke-width: 1
    fill-graphic:
      symbols:
      - mark:
          shape: square
          fill-color: 'gray'
          stroke-color: 'black'
          stroke-width: 1
```

4. To outline the individual shapes:



5. Additional fill properties allow control over the orientation and size of the symbol.

The **size** property is used to adjust the size of the symbol prior to use.

The **rotation** property is used to adjust the orientation of the symbol.

Adjust the size and rotation as shown:

```
symbolizers:
- polygon:
    stroke-color: 'black'
    stroke-width: 1
    fill-graphic:
      size: 22
      rotation: 45.0
      symbols:
      - mark:
          shape: square
          fill-color: 'gray'
          stroke-color: 'black'
          stroke-width: 1
```

6. The size of each symbol is increased, and each symbol rotated by 45 degrees.

Note: Does the above look correct? There is an open request GEOT-4642 to rotate the entire pattern, rather than each individual symbol.

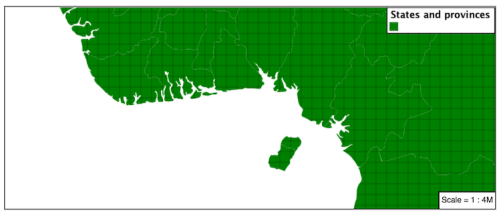
7. The size and rotation properties just affect the size and placement of the symbol, but do not alter the symbol's design. In order to control the color we set the **fill-color** and **stroke-color** properties of the **mark**.

8. Replace the contents of `polygon_example` with the following:

```
symbolizers:
- polygon:
    fill-graphic:
      symbols:
      - mark:
          shape: square
          fill-color: '#008000'
          stroke-color: '#006400'
```

```
        stroke-width: 1
```

9. This change adjusts the appearance of our grid of squares.



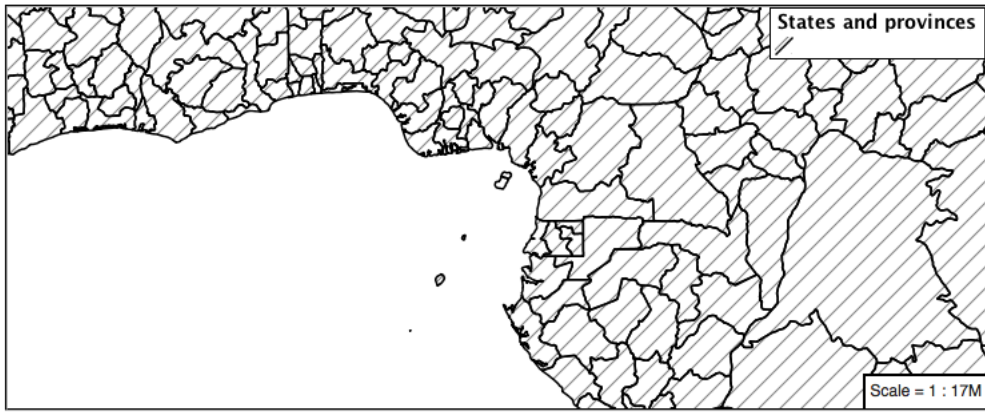
10. The well-known symbols are more suited for marking individual points. Now that we understand how a pattern can be controlled it is time to look at the patterns GeoServer provides.

shape://horizline	horizontal hatching
shape://vertline	vertical hatching
shape://backslash	right hatching pattern
shape://slash	left hatching pattern
shape://plus	vertical and horizontal hatching pattern
shape://times	cross hatch pattern

Update the example to use **shape://slash** for a pattern of left hatching.

```
symbolizers:
- polygon:
    stroke-color: 'black'
    stroke-width: 1
    fill-graphic:
      symbols:
      - mark:
          shape: 'shape://slash'
          stroke-color: 'gray'
```

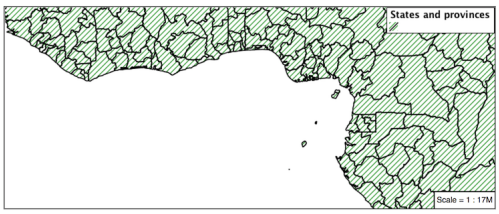
11. This approach is well suited to printed output or low color devices.



12. To control the size of the symbol produced use the **size** property of the **fill-graphic**.

```
symbolizers:
- polygon:
    stroke-color: 'black'
    stroke-width: 1
    fill-graphic:
      size: 8
      symbols:
      - mark:
          shape: 'shape://slash'
          stroke-color: 'gray'
```

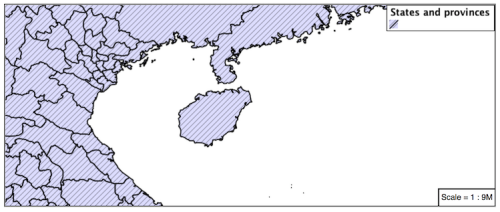
13. This results in a tighter pattern shown:



14. Multiple fills can be applied by using a seperate symbolizer for each fill as part of the same rule.

```
symbolizers:
- polygon:
    stroke-color: 'black'
    stroke-width: 1
    fill-color: '#DDDDFF'
- polygon:
    fill-graphic:
      size: 8
      symbols:
      - mark:
          shape: shape://slash
          stroke-color: 'black'
          stroke-width: 0.5
```

15. The resulting image has a solid fill, with a pattern drawn overtop.



Label

Labeling polygons follows the same approach used for LineStrings.

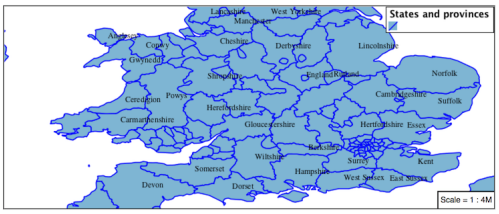
The key properties **fill** and **label** are used to enable Polygon label generation.

1. By default labels are drawn starting at the centroid of each polygon.

2. Try out **label** and **fill** together by replacing our `polygon_example` with the following:

```
symbolizers:
- polygon:
    stroke-color: 'blue'
    stroke-width: 1
    fill-color: '#7EB5D3'
- text:
    label: ${name}
    fill-color: 'black'
```

3. Each label is drawn from the lower-left corner as shown in the `Layer Preview` preview.



4. We can adjust how the label is drawn at the polygon centroid.

   The property **anchor** provides two numbers expressing how a label is aligned with respect to the centroid. The first value controls the horizontal alignment, while the second value controls the vertical alignment. Alignment is expressed between 0.0 and 1.0 as shown in the following table.

	Left	Center	Right
Top	0.0 1.0	0.5 1.0	1.0 1.0
Middle	0.0 0.5	0.5 0.5	1.0 0.5
Bottom	0.0 0.0	0.5 0.0	1.0 0.0

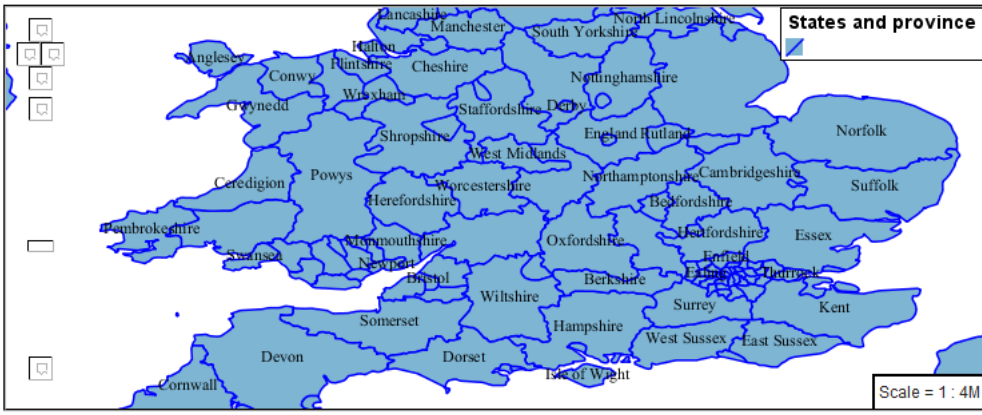
   Adjusting the **anchor** is the recommended approach to positioning your labels.

5. Using the **anchor** property we can center our labels with respect to geometry centroid.

   To align the center of our label we select 50% horizontally and 50% vertically, by filling in 0.5 and 0.5 below:

```
symbolizers:
- polygon:
    stroke-color: 'blue'
    stroke-width: 1
    fill-color: '#7EB5D3'
- text:
    label: ${name}
    fill-color: 'black'
    anchor: [0.5, 0.5]
```

6. The labeling position remains at the polygon centroid. We adjust alignment by controlling which part of the label we are "snapping" into position.

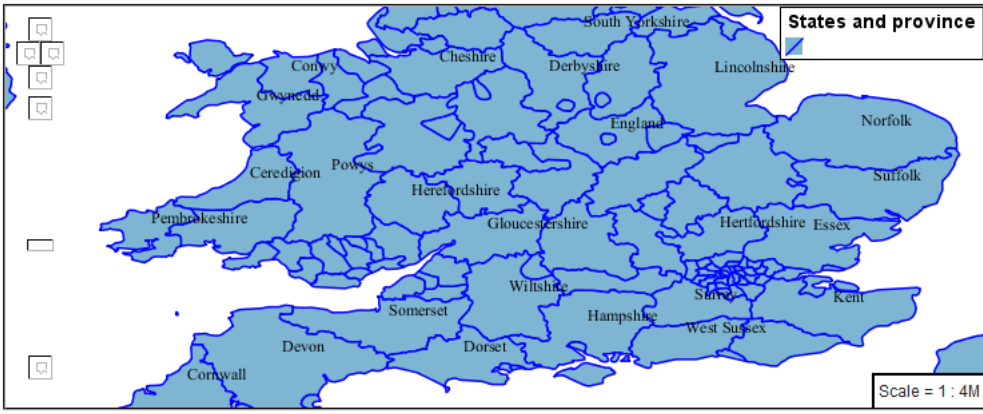


7. The property **displacement** can be used to provide an initial displacement using and x and y offset.

8. This offset is used to adjust the label position relative to the geometry centroid resulting in the starting label position.

```
symbolizers:
- polygon:
    stroke-color: 'blue'
    stroke-width: 1
    fill-color: '#7EB5D3'
- text:
    label: ${name}
    fill-color: 'black'
    displacement: [0, 7]
```

9. Confirm this result in the map preview.



10. These two settings can be used together.

The rendering engine starts by determining the label position generated from the geometry centroid and the **label-offset** displacement.

The bounding box of the label is used with the **label-anchor** setting align the label to this location.

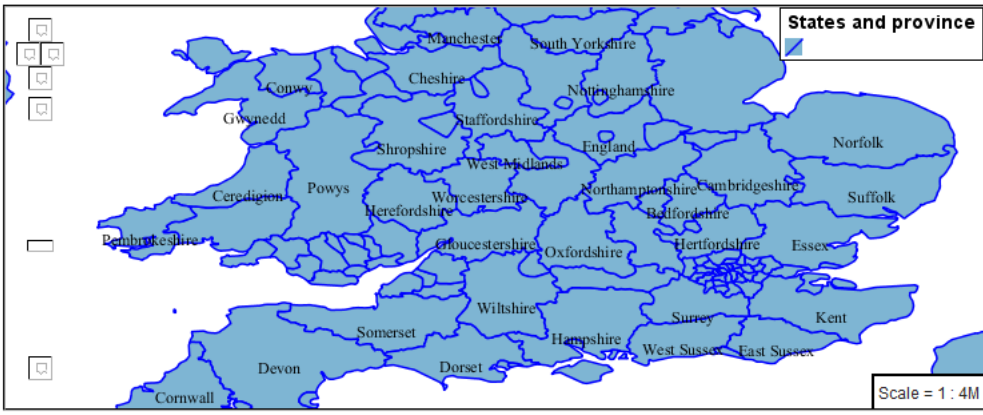
Step 1: starting label position = centroid + displacement

Step 2: snap the label anchor to the starting label position

11. To move our labels down (allowing readers to focus on each shape) we can use displacement combined with followed by horizontal alignment.

```
symbolizers:
- polygon:
    stroke-color: 'blue'
    stroke-width: 1
    fill-color: '#7EB5D3'
- text:
    label: ${name}
    fill-color: 'black'
    anchor: [0.5, 1]
    displacement: [0, -7]
```

12. As shown in the map preview.



Legibility

When working with labels a map can become busy very quickly, and difficult to read.

1. GeoServer provides extensive vendor parameters directly controlling the labelling process.

   Many of these parameters focus on controlling conflict resolution (when labels would otherwise overlap).

2. Two common properties for controlling labeling are:

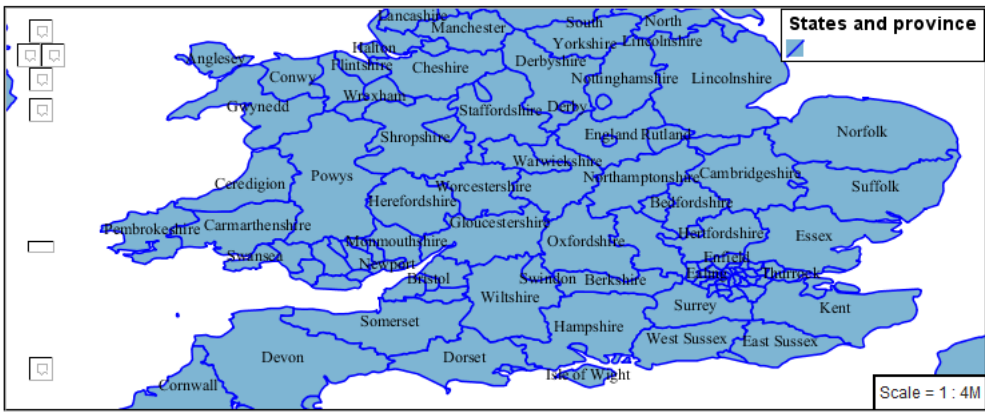
   x-maxDisplacement indicates the maximum distance GeoServer should displace a label during conflict resolution.

   x-autoWrap allows any labels extending past the provided width will be wrapped into multiple lines.

3. Using these together we can make a small improvement in our example:

```
symbolizers:
- polygon:
    stroke-color: 'blue'
    stroke-width: 1
    fill-color: '#7EB5D3'
- text:
    label: ${name}
    fill-color: 'black'
    anchor: [0.5, 0.5]
    x-maxDisplacement: 40
    x-autoWrap: 70
```

4. As shown in the following preview.



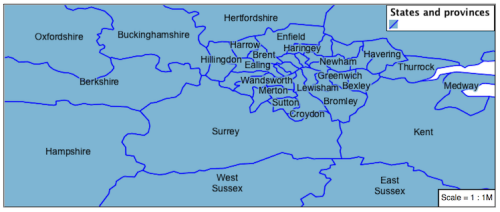
5. Even with this improved spacing between labels, it is difficult to read the result against the complicated line work.

   Use of a halo to outline labels allows the text to stand out from an otherwise busy background. In this case we will make use of the fill color, to provide some space around our labels. We will also change the font to Arial.

```
symbolizers:
- polygon:
    stroke-color: 'blue'
    stroke-width: 1
    fill-color: '#7EB5D3'
- text:
    label: ${name}
    fill-color: 'black'
    anchor: [0.5, 0.5]
    font-family: Arial
    font-size: 14
    font-style: normal
    font-weight: normal
    halo:
      fill-color: '#7EB5D3'
      fill-opacity: 0.8
      radius: 2
    x-maxDisplacement: 40
    x-autoWrap: 70
```

6. By making use of **fill-opacity** on the **halo** we we still allow stroke

information to show through, but prevent the stroke information from making the text hard to read.



7. And advanced technique for manually taking control of conflict resolution is the use of the **priority**.

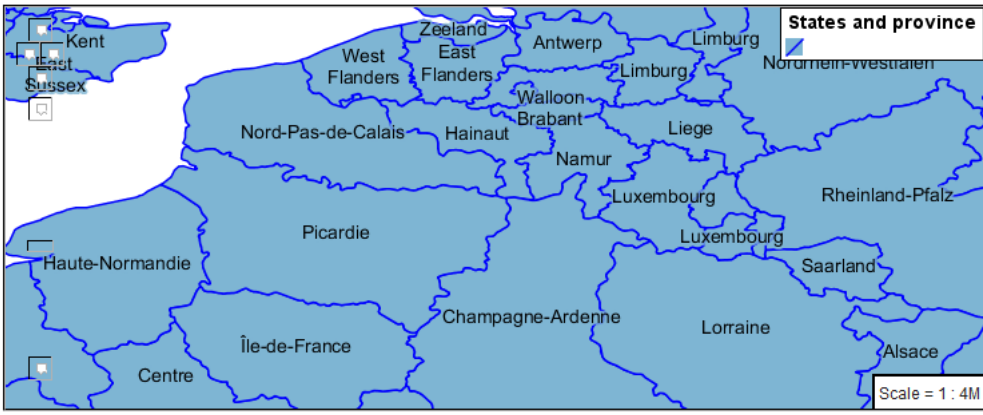
   This property takes an expression which is used in the event of a conflict. The label with the highest priority "wins."

8. The Natural Earth dataset we are using includes a **labelrank** intended to control what labels are displayed based on zoom level.

   The values for **labelrank** go from 0 (for zoomed out) to 20 (for zoomed in). To use this value for **priority** we need to swap the values around so a **scalerank** of 1 is given the highest priority.

```
symbolizers:
- polygon:
    stroke-color: 'blue'
    stroke-width: 1
    fill-color: '#7EB5D3'
- text:
    label: ${name}
    fill-color: 'black'
    anchor: [0.5, 0.5]
    font-family: Arial
    font-size: 14
    font-style: normal
    font-weight: normal
    halo:
      fill-color: '#7EB5D3'
      fill-opacity: 0.8
      radius: 2
    x-maxDisplacement: 40
    x-autoWrap: 70
    priority: ${'20' - labelrank}
```

9. In the following map `East Flanders` will take priority over `Zeeland` when the two labels overlap.



Theme

A thematic map (rather than focusing on representing the shape of the world) uses elements of style to illustrate differences in the data under study. This section is a little more advanced and we will take the time to look at the generated SLD file.

1. We can use a site like ColorBrewer to explore the use of color theming for polygon symbology. In this approach the the fill color of the polygon is determined by the value of the attribute under study.

   This presentation of a dataset is known as "theming" by an attribute.

2. For our `ne:states_provinces_shp` dataset, a **mapcolor9** attribute has been provided for this purpose. Theming by **mapcolor9** results in a map where neighbouring countries are visually distinct.

Qualitative 9-class Set3		
#8dd3c7	#fb8072	#b3de69
#ffffb3	#80b1d3	#fccde5
#bebada	#fdb462	#d9d9d9

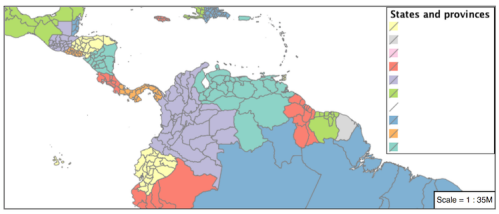
   If you are unfamiliar with theming you may wish to visit http://colorbrewer2.org to learn more. The **i** icons provide an adequate background on theming approaches for qualitative, sequential and diverging datasets.

3. The first approach we will take is to directly select content based on **colormap**, providing a color based on the **9-class Set3** palette above:

```
define: &stroke
  stroke-color: 'gray'
  stroke-width: 0.5
rules:
  - filter: ${mapcolor9 = '1'}
    scale: [min, max]
    symbolizers:
    - polygon:
        <<: *stroke
```

```
            fill-color: '#8DD3C7'
      - filter: ${mapcolor9 = '2'}
        scale: [min, max]
        symbolizers:
        - polygon:
            <<: *stroke
            fill-color: '#FFFFB3'
      - filter: ${mapcolor9 = '3'}
        scale: [min, max]
        symbolizers:
        - polygon:
            <<: *stroke
            fill-color: '#BEBADA'
      - filter: ${mapcolor9 = '4'}
        scale: [min, max]
        symbolizers:
        - polygon:
            <<: *stroke
            fill-color: '#FB8072'
      - filter: ${mapcolor9 = '5'}
        scale: [min, max]
        symbolizers:
        - polygon:
            <<: *stroke
            fill-color: '#80B1D3'
      - filter: ${mapcolor9 = '6'}
        scale: [min, max]
        symbolizers:
        - polygon:
            <<: *stroke
            fill-color: '#FDB462'
      - filter: ${mapcolor9 = '7'}
        scale: [min, max]
        symbolizers:
        - polygon:
            <<: *stroke
            fill-color: '#B3DE69'
      - filter: ${mapcolor9 = '8'}
        scale: [min, max]
        symbolizers:
        - polygon:
            <<: *stroke
            fill-color: '#FCCDE5'
      - filter: ${mapcolor9 = '9'}
        scale: [min, max]
        symbolizers:
        - polygon:
            <<: *stroke
            fill-color: '#D9D9D9'
      - filter: ${mapcolor9 <> '1' AND mapcolor9␣
    ↪<> '2' AND mapcolor9 <> '3' AND mapcolor9 <> '4' AND␣
    ↪mapcolor9 <> '5' AND mapcolor9 <> '6' AND mapcolor9␣
    ↪<> '7' AND mapcolor9 <> '8' AND mapcolor9 <> '9'}
        scale: [min, max]
        symbolizers:
        - line:
            <<: *stroke
```

4. The *Layer Preview* tab can be used to preview this result.



5. This YSLD makes use of a **define** to avoid repeating the **stroke-color** and **stroke-width** information multiple times.

   As an example the `${mapcolor9 = '2'}` rule, combined with the `define:` results in the following collection of properties:

```
- filter: ${mapcolor9 = '2'}
    scale: [min, max]
    symbolizers:
    - polygon:
        stroke-color: 'gray'
        stroke-width: 0.5
        fill-color: '#FFFFB3'
```

6. Reviewing the generated SLD shows us this representation:

```
<sld:Rule>
   <ogc:Filter>
      <ogc:PropertyIsEqualTo>
         <ogc:PropertyName>mapcolor9</ogc:PropertyName>
         <ogc:Literal>2</ogc:Literal>
      </ogc:PropertyIsEqualTo>
   </ogc:Filter>
   <sld:PolygonSymbolizer>
      <sld:Fill>
         <sld:CssParameter␣
↪name="fill">#ffffb3</sld:CssParameter>
      </sld:Fill>
   </sld:PolygonSymbolizer>
   <sld:LineSymbolizer>
      <sld:Stroke>
         <sld:CssParameter␣
↪name="stroke">#808080</sld:CssParameter>
         <sld:CssParameter␣
↪name="stroke-width">0.5</sld:CssParameter>
      </sld:Stroke>
   </sld:LineSymbolizer>
</sld:Rule>
```

7. There are three important functions, defined by the Symbology Encoding specification, that are often easier to use for theming than using rules.

- **Recode**: Used the theme qualitative data. Attribute values are directly mapped to styling property such as **fill** or **stroke-width**.
- **Categorize**: Used the theme quantitative data. Categories are defined using min and max ranges, and values are sorted into the appropriate category.

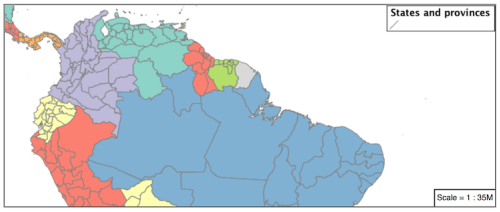
- **Interpolate**: Used to smoothly theme quantitative data by calculating a styling property based on an attribute value.

  Theming is an activity, producing a visual result allow map readers to learn more about how an attribute is distributed spatially. We are free to produce this visual in the most efficient way possible.

8. Swap out **mapcolor9** theme to use the **Recode** function:

```
symbolizers:
- polygon:
    stroke-color: 'gray'
    stroke-width: 0.5
    fill-color: ${Recode(mapcolor9,
      '1','#8dd3c7',
      '2','#ffffb3',
      '3','#bebada',
      '4','#fb8072',
      '5','#80b1d3',
      '6','#fdb462',
      '7','#b3de69',
      '8','#fccde5',
      '9','#d9d9d9')}
```

9. The *Layer Preview* tab provides the same preview.



10. The *Generated SLD* tab shows where things get interesting. Our generated style now consists of a single **Rule**:

```
<sld:Rule>
   <sld:PolygonSymbolizer>
      <sld:Fill>
         <sld:CssParameter name="fill">
            <ogc:Function name="Recode">
↪
↪        <ogc:PropertyName>mapcolor9</ogc:PropertyName>
               <ogc:Literal>1</ogc:Literal>
                  <ogc:Literal>#8dd3c7</ogc:Literal>
               <ogc:Literal>2</ogc:Literal>
                  <ogc:Literal>#ffffb3</ogc:Literal>
               <ogc:Literal>3</ogc:Literal>
                  <ogc:Literal>#bebada</ogc:Literal>
               <ogc:Literal>4</ogc:Literal>
                  <ogc:Literal>#fb8072</ogc:Literal>
               <ogc:Literal>5</ogc:Literal>
                  <ogc:Literal>#80b1d3</ogc:Literal>
               <ogc:Literal>6</ogc:Literal>
                  <ogc:Literal>#fdb462</ogc:Literal>
               <ogc:Literal>7</ogc:Literal>
                  <ogc:Literal>#b3de69</ogc:Literal>
               <ogc:Literal>8</ogc:Literal>
```

```
                                        <ogc:Literal>#fccde5</ogc:Literal>
                                    <ogc:Literal>9</ogc:Literal>
                                        <ogc:Literal>#d9d9d9</ogc:Literal>
                                </ogc:Function>
                                </sld:CssParameter>
                            </sld:Fill>
                        </sld:PolygonSymbolizer>
                        <sld:LineSymbolizer>
                            <sld:Stroke>
                                <sld:CssParameter␣
↪name="stroke">#808080</sld:CssParameter>
                                <sld:CssParameter␣
↪name="stroke-width">0.5</sld:CssParameter>
                            </sld:Stroke>
                        </sld:LineSymbolizer>
                    </sld:Rule>
```

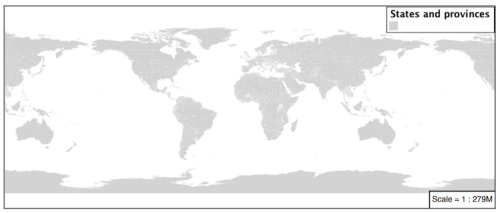
Bonus

The following optional explore and challenge activities offer a chance to review and apply the ideas introduced here. The challenge activities equire a bit of creativity and research to complete.

In a classroom setting you are encouraged to team up into groups, with each group taking on a different challenge.

Explore Antialiasing

1. When we rendered our initial preview, without a stroke, thin white gaps (or slivers) are visible between our polygons.

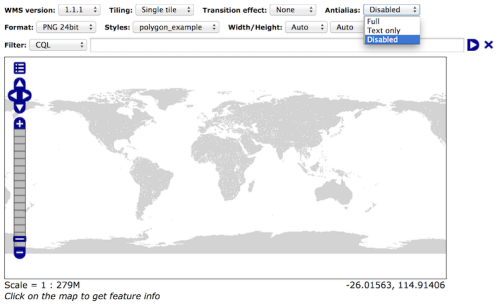


   This effect is made more pronounced by the rendering engine making use of the Java 2D sub-pixel accuracy. This technique is primarily used to prevent an aliased (stair-stepped) appearance on diagonal lines.

2. Clients can turn this feature off using a GetMap format option:

```
format_options=antialiasing=off;
```

   The **LayerPreview** provides access to this setting from the Open Layers **Options Toolbar**:



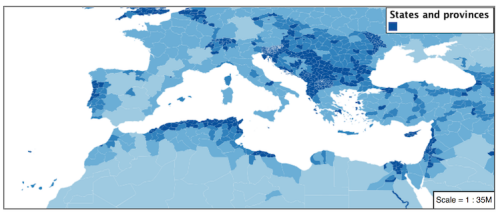
3. **Explore:** Experiment with **fill** and **stroke** settings to eliminate slivers between polygons.

Note: Answer *provided* at the end of the workbook.

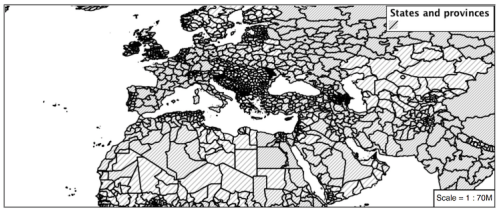
Explore Categorize

1. The **Categorize** function can be used to generate property values based on quantitative information. Here is an example using Categorize to color states according to size.

```
symbolizers:
- polygon:
    fill-color: ${Categorize(Shape_Area,
      '#08519c','0.5',
      '#3182bd','1',
      '#6baed6','5',
      '#9ecae1','60',
      '#c6dbef','80',
      '#eff3ff')}
```



2. An exciting use of the GeoServer **shape** symbols is the theming by changing the **size** used for pattern density.
3. **Explore:** Use the **Categorize** function to theme by **datarank**.



Note: Answer *provided* at the end of the workbook.

Challenge Goodness of Fit

1. A subject we touched on during labeling was the conflict resolution GeoServer performs to ensure labels do not overlap.
2. In addition to the vendor parameter for max displacement you can experiment with different values for "goodness of fit". These settings control how far GeoServer is willing to move a label to avoid conflict, and under what terms it simply gives up:

```
x-goodnessOfFit: 0.3
x-maxDisplacement: 130
```

3. You can also experiment with turning off this facility completely:

```
x-conflictResolution: false
```

4. **Challenge:** Construct your own example using maxDisplacement and goodnessOfFit.

Challenge Halo

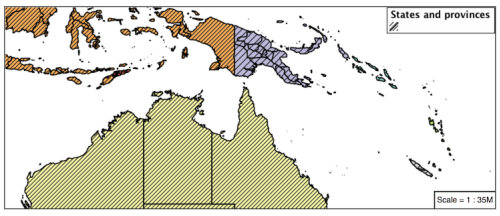
1. The halo example used the fill color and opacity for a muted halo, while this improved readability it did not bring attention to our labels.

   A common design choice for emphasis is to outline the text in a contrasting color.
2. **Challenge:** Produce a map that uses a white halo around black text.

Note: Answer *provided* at the end of the workbook.

Challenge Theming using Multiple Attributes

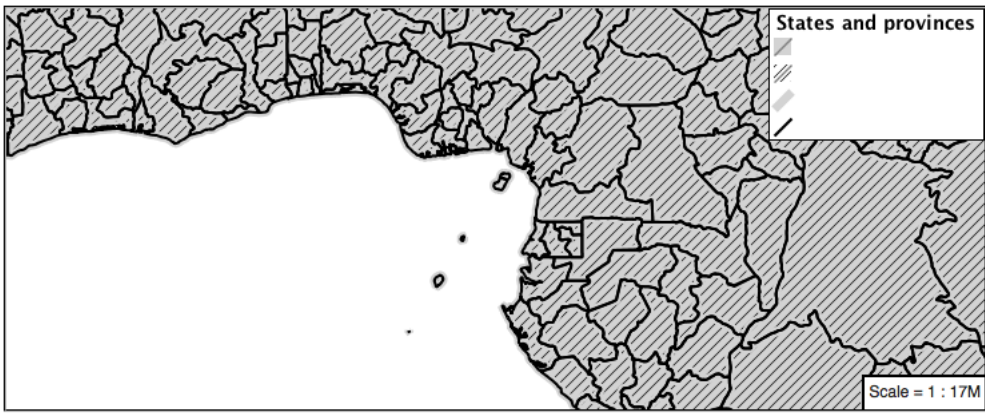
1. A powerful tool is theming using multiple attributes. This is an important concept allowing map readers to perform "integration by eyeball" (detecting correlations between attribute values information).
2. **Challenge:** Combine the **mapcolor9** and **datarank** examples to reproduce the following map.



Note: Answer *provided* at the end of the workbook.

Challenge Use of Z-Index

1. Earlier we looked at using multiple **feature-styles** to simulate line string casing. The line work was drawn twice, once with thick line, and then a second time with a thinner line. The resulting effect is similar to text halos - providing breathing space around complex line work allowing it to stand out.
2. **Challenge:** Use what you know of LineString **feature-styles** to reproduce the following map:



Note: Answer *provided* at the end of the workbook.

Points

The next stop of the ysld styling tour is the representation of points.

Review of point symbology:

- Points are used to represent a location only, and do not form a shape. The visual width of lines do not change depending on scale.
- SLD uses a **PointSymbolizer** record how the shape of a line is drawn.
- Labeling of points is anchored to the point location.

  As points have no inherent shape of of their own, emphasis is placed on marking locations with an appropriate symbol.

Reference:

- *YSLD Reference*
- *YSLD Reference Point symbolizer* (User Manual | YSLD Reference)
- *Point* (User Manual | SLD Reference)

This exercise makes use of the `ne:populated_places` layer.

1. Navigate to the **Styles** page.

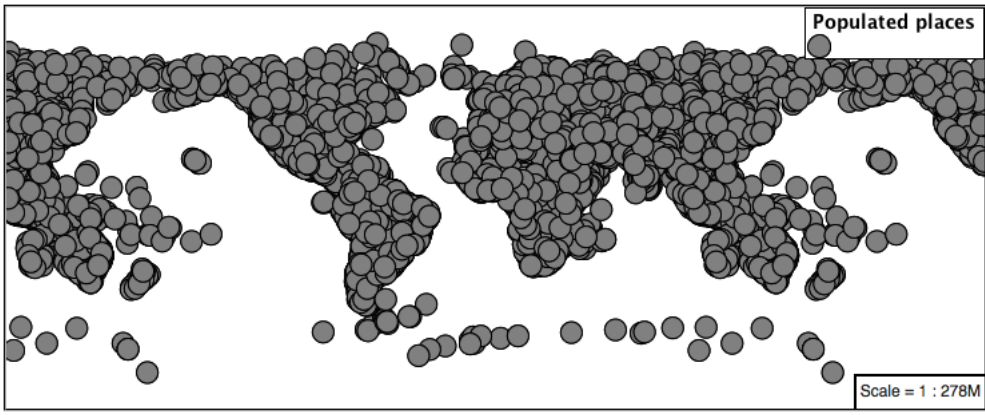
2. Click *Add a new style* and choose the following:

Name:	`point_example`
Workspace:	`No workspace`
Format:	`YSLD`

3. Choose *point* from the `Generate a default style` dropdown and click *generate*.
4. Replace the initial YSLD definition with the following and click *apply*:

```
symbolizers:
- point:
    symbols:
    - mark:
        shape: circle
        stroke-width: 1
```

5. And use the *Layer Preview* tab to preview the result.



Mark

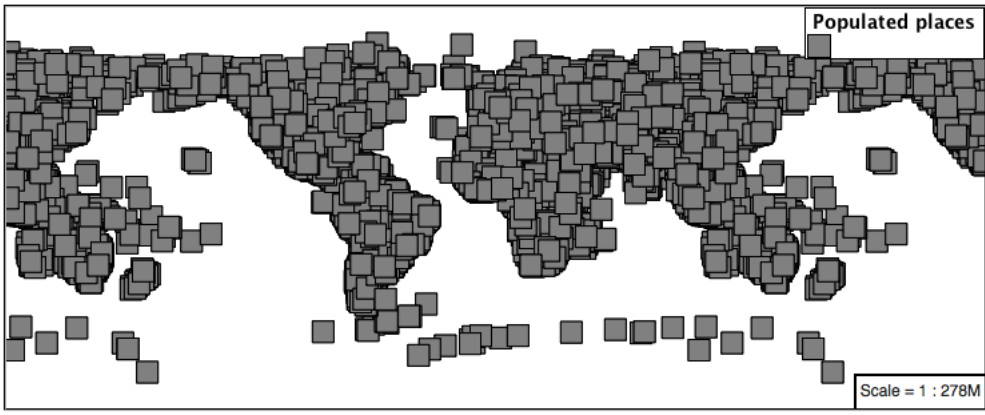
The **point** symbolizer controls the display of point data. Points are represented with the mandatory property **mark**.

The SLD standard provides "well-known" symbols for use with point symbology: `circle`, `square`, `triangle`, `arrow`, `cross`, `star`, and `x`.

1. Change the symbol used by the style to a square:

```
symbolizers:
- point:
    symbols:
    - mark:
        shape: square
        stroke-width: 1
```

2. Map Preview:

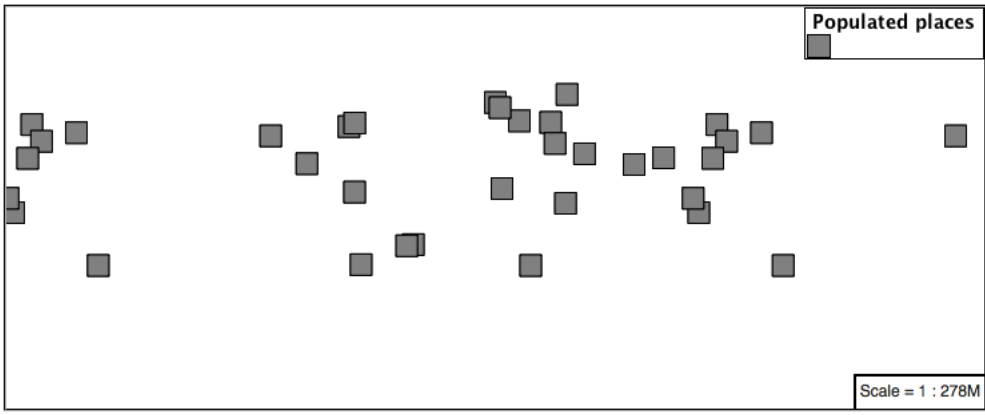


3. Before we continue we will use a filter to cut down the amount of data shown to a reasonable level.

```
rules:
- filter: ${SCALERANK < '1'}
  scale: [min, max]
  symbolizers:
  - point:
      symbols:
      - mark:
          shape: square
          stroke-width: 1
```

Note: **symbolizers** has been indented under **rules**

1. Resulting in a considerably cleaner image:



2. Additional properties are available to control a mark's presentation:

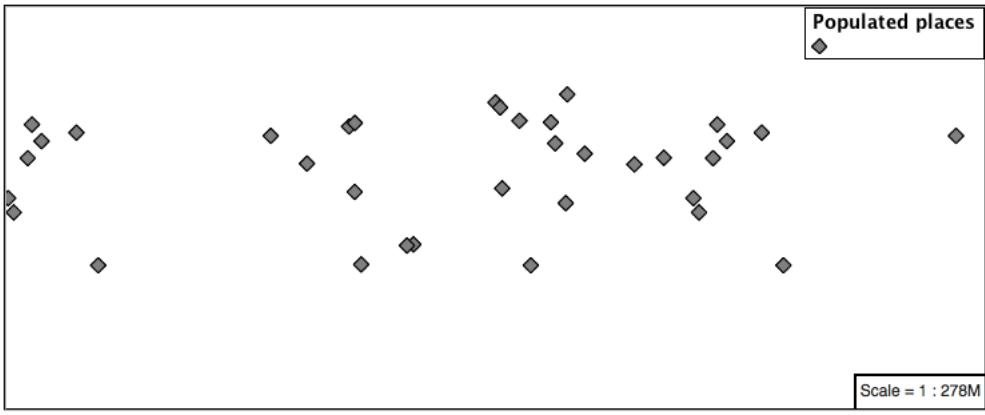
   The **size** property is used to control symbol size.

   The **rotation** property controls orientation, accepting input in degrees.

   Trying these two settings together:

```
rules:
- filter: ${SCALERANK < '1'}
  scale: [min, max]
  symbolizers:
  - point:
      size: 8
      rotation: 45.0
      symbols:
      - mark:
          shape: square
          stroke-width: 1
```

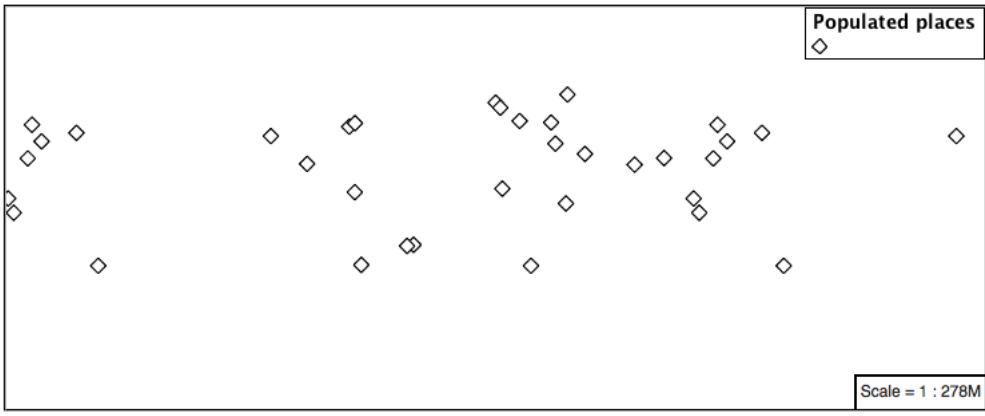
3. Results in each location being marked with a diamond:



4. The **mark** property provides parameters to style the point symbol. Let's change the **fill-color** to gray.

```
rules:
- filter: ${SCALERANK < '1'}
  scale: [min, max]
  symbolizers:
  - point:
      size: 8
      rotation: 45.0
      symbols:
      - mark:
          shape: square
          stroke-color: 'black'
          stroke-width: 1
          fill-color: 'gray'
```

5. Updating the mark to a gray square with a black outline.

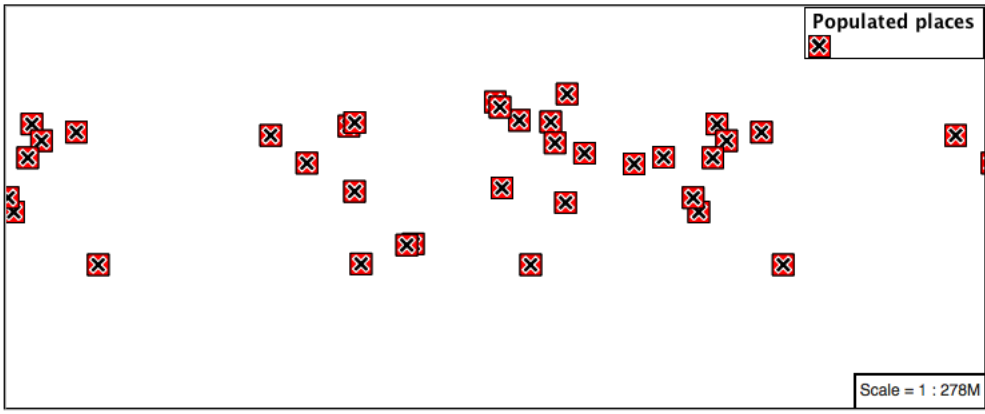


6. You can add more symbolizers to apply additional point styles.

   Using this approach marks can be composed of multiple symbols, each with its own settings:

```
rules:
- filter: ${SCALERANK < '1'}
  scale: [min, max]
  symbolizers:
  - point:
      size: 16
      symbols:
      - mark:
          shape: square
          stroke-color: 'black'
          stroke-width: 1
          fill-color: 'red'
  - point:
      size: 14
      rotation: 45.0
      symbols:
      - mark:
          shape: cross
          stroke-color: 'white'
          stroke-width: 1
          fill-color: 'black'
```

7. Producing an interesting compound symbol effect:



Graphic

Symbols can also be supplied by an external graphic,

This technique was shown with the initial `airport.svg` YSLD example.

1. To use an external graphic two pieces of information are required.

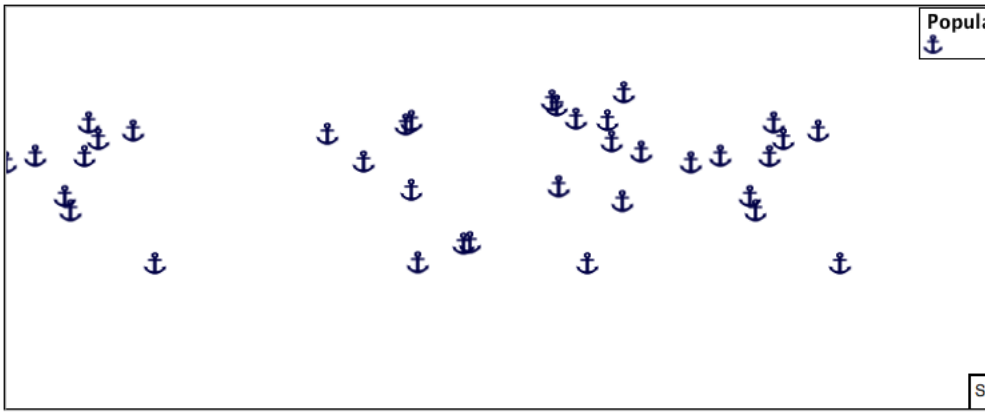
   url property is defined with a **url** reference to image.

   format property is used to tell the rendering engine what file format to expect

   This technique is used to reference files placed in the styles directory.

```
rules:
- filter: ${SCALERANK < '1'}
  scale: [min, max]
  symbolizers:
  - point:
      symbols:
      - external:
          url: file:/path/to/geoserver/data_dir/styles/port.svg
          format: image/svg
```

2. Drawing the provided shape in each location:

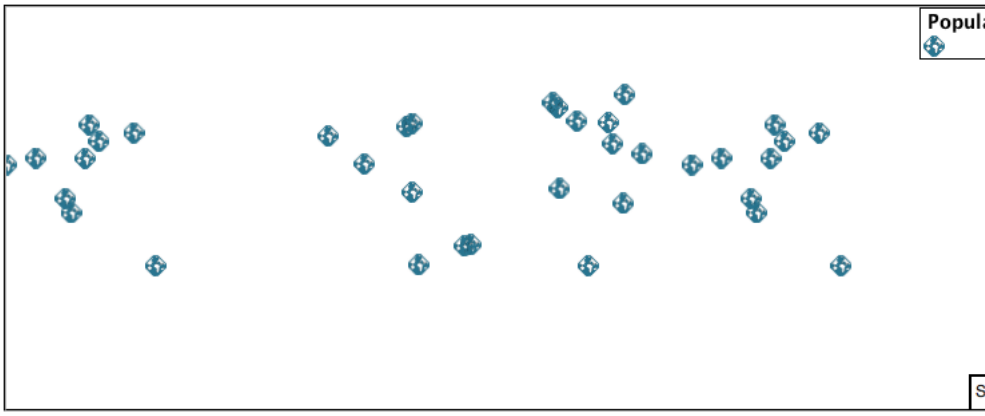


3. The property **url** reference can also be used to reference external images. We can make use of the GeoServer logo.

```
rules:
- filter: ${SCALERANK < '1'}
  scale: [min, max]
  symbolizers:
  - point:
      size: 16
      symbols:
      - external:
          url: http://localhost:8080/geoserver/web/wicket/resource/org.geoserver.web.GeoServerBasePage/img/logo.png
```

```
format: image/png
```

4. As shown in the map preview.



Label

Labeling is now familiar from our experience with LineString and Polygons.

The **text** symbolizer with the **label** property are required to label Point Locations.

1. Replace `point_example` with the following:

```
rules:
- filter: ${SCALERANK < '1'}
  scale: [min, max]
  symbolizers:
  - point:
      symbols:
      - mark:
          shape: circle
          stroke-color: 'black'
          stroke-width: 1
          fill-color: 'gray'
  - text:
      label: ${NAME}
      fill-color: 'gray'
      placement: point
```

2. Confirm the result in `Layer Preview` preview.

3. Each label is drawn starting from the provided point - which is unfortunate as it assures each label will overlap with the symbol used. To fix this limitation we will make use of the YSLD controls for label placement:

   anchor provides two values expressing how a label is aligned with respect to the starting label position.

   displacement is be used to provide an initial displacement using and x and y offset. For points this offset is recommended to adjust the label position away for the area used by the symbol.

   Note: The property **anchor** defines an anchor position relative to the bounding box formed by the resulting label. This anchor position is snapped to the label position generated by the point location and displacement offset.

   Using these two facilities together we can center our labels below the symbol, taking care that the displacement used provides an offset just outside the area required for the symbol size.

```
rules:
- filter: ${SCALERANK < '1'}
  scale: [min, max]
  symbolizers:
  - point:
      size: 10
      symbols:
      - mark:
          shape: circle
          stroke-color: 'black'
          stroke-width: 1
          fill-color: 'gray'
  - text:
      label: ${NAME}
      fill-color: 'black'
      placement: point
      anchor: [0.5, 1.0]
      displacement: [0, -12]
```

4. Each label is now placed under the mark.

5. One remaining issue is the overlap between labels and symbols.

   GeoServer provides a vendor specific parameter to allow symbols to take part in label conflict resolution, preventing labels from overlapping any symbols. This severely limits the area available for labeling and is best used in conjunction with a large maximum displacement vendor option.

   x-labelObstacle vendor parameter asks the rendering engine to avoid drawing labels over top of the indicated symbol. This applies to the point symbolizer.

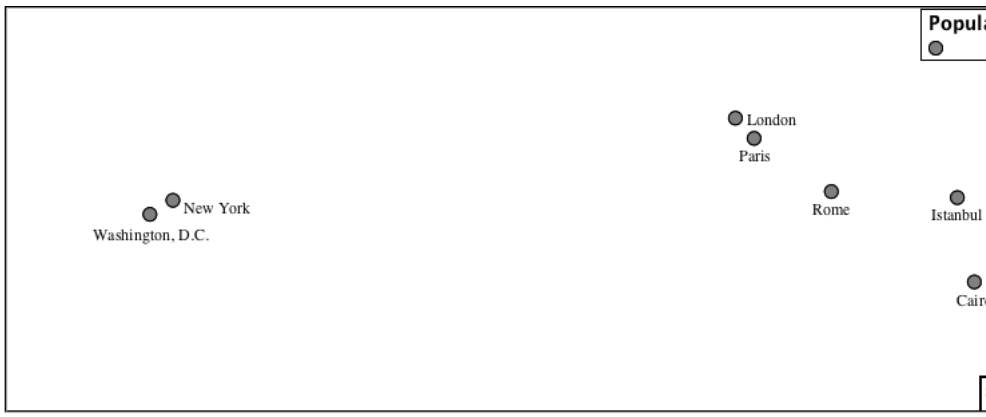
   x-maxDisplacement vendor parameter provides the rendering engine a maximum distance it is allowed to move labels during conflict resolution. This applies to the text symbolizer.

   x-spaceAround vendor parameter tells the rendering engine to provide a minimum distance between the labels on the map, ensuring they do not overlap. This applies to the text symbolizer.

   Update our example to use these settings:

```
rules:
- filter: ${SCALERANK < '1'}
  scale: [min, max]
  symbolizers:
  - point:
      size: 10
      symbols:
      - mark:
          shape: circle
          stroke-color: 'black'
          stroke-width: 1
          fill-color: 'gray'
      x-labelObstacle: true
  - text:
      label: ${NAME}
      fill-color: 'black'
      placement: point
      anchor: [0.5, 1.0]
      displacement: [0, -12]
      x-maxDisplacement: 100
      x-spaceAround: 2
```

6. Resulting in a considerably cleaner image:



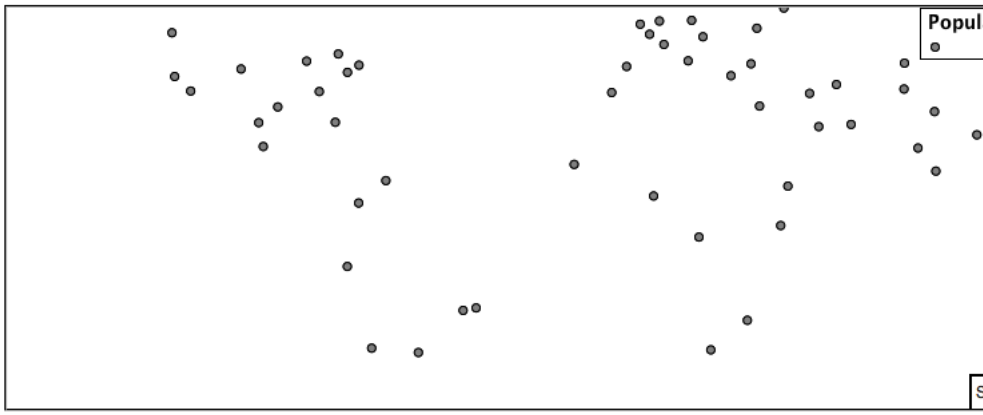
Dynamic Styling

1. We will quickly use **scalerank** to select content based on @scale filters.

```
define: &point
  size: 6
  symbols:
  - mark:
      shape: circle
      stroke-color: 'black'
      stroke-width: 1
      fill-color: 'gray'
rules:
- filter: ${SCALERANK < '7'}
  scale: ['4000000.0', '8000000.0']
  symbolizers:
  - point:
      <<: *point
- filter: ${SCALERANK < '5'}
  scale: ['8000000.0', '1.7E7']
  symbolizers:
  - point:
      <<: *point
- filter: ${SCALERANK < '4'}
  scale: ['1.7E7', '3.5E7']
  symbolizers:
  - point:
      <<: *point
- filter: ${SCALERANK < '3'}
  scale: ['3.5E7', '7.0E7']
  symbolizers:
  - point:
      <<: *point
- filter: ${SCALERANK < '2'}
  scale: ['7.0E7', '1.4E8']
  symbolizers:
  - point:
      <<: *point
- filter: ${SCALERANK < '1'}
  scale: ['1.4E8', max]
  symbolizers:
```

```
    - point:
        <<: *point
  - scale: [min, '4000000.0']
    symbolizers:
    - point:
        <<: *point
```

2. Click *Apply* to update the *Layer Preview* after each step.



Note: This YSLD makes use of a **define** to avoid repeating the point symbolizer content multiple times. As an example the `scale: [min, '4000000.0']` rule, combined with the `define:` results in the following collection of properties:

```
  - scale: [min, '4000000.0']
    symbolizers:
    - point:
        size: 6
        symbols:
        - mark:
            shape: circle
            stroke-color: 'black'
            stroke-width: 1
            fill-color: 'gray'
```

3. To add labeling we must use both a point and text symbolizer in each scale filter.

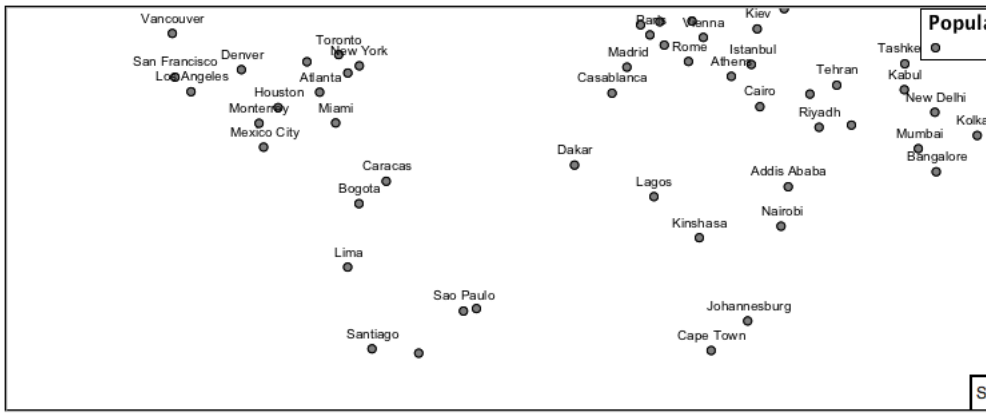
```
define: &point
  size: 6
  symbols:
  - mark:
      shape: circle
      stroke-color: 'black'
      stroke-width: 1
      fill-color: 'gray'
define: &label
  label: ${NAME}
  fill-color: 'black'
  font-family: Arial
  font-size: 10
```

```
      font-style: normal
      font-weight: normal
      placement: point
    rules:
    - filter: ${SCALERANK < '7'}
      scale: ['4000000.0', '8000000.0']
      symbolizers:
      - point:
          <<: *point
      - text:
          <<: *label
    - filter: ${SCALERANK < '5'}
      scale: ['8000000.0', '1.7E7']
      symbolizers:
      - point:
          <<: *point
      - text:
          <<: *label
    - filter: ${SCALERANK < '4'}
      scale: ['1.7E7', '3.5E7']
      symbolizers:
      - point:
          <<: *point
      - text:
          <<: *label
    - filter: ${SCALERANK < '3'}
      scale: ['3.5E7', '7.0E7']
      symbolizers:
      - point:
          <<: *point
      - text:
          <<: *label
    - filter: ${SCALERANK < '2'}
      scale: ['7.0E7', '1.4E8']
      symbolizers:
      - point:
          <<: *point
      - text:
          <<: *label
    - filter: ${SCALERANK < '1'}
      scale: ['1.4E8', max]
      symbolizers:
      - point:
          <<: *point
      - text:
          <<: *label
    - scale: [min, '4000000.0']
      symbolizers:
      - point:
          <<: *point
      - text:
          <<: *label
```

4. We will use **displacement** and **anchor** to position the label above each symbol.

   Add the following two lines to the `label` define:

```
define: &label
  label: ${NAME}
  fill-color: 'black'
  font-family: Arial
  font-size: 10
  font-style: normal
  font-weight: normal
  placement: point
  anchor: [0.5, 0]
  displacement: [0, 6]
```



5. A little bit of work with vendor specific parameters will prevent our labels from colliding with each symbol, while giving the rendering engine some flexibility in how far it is allowed to relocate a label.

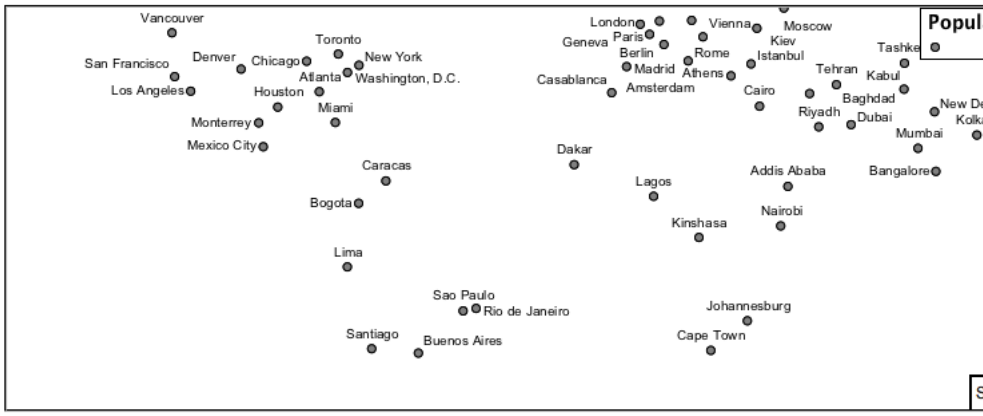
   Add the following vendor options to the `label` define:

```
define: &label
  label: ${NAME}
  fill-color: 'black'
  font-family: Arial
  font-size: 10
  font-style: normal
  font-weight: normal
  placement: point
```

```
    anchor: [0.5, 0]
    displacement: [0, 6]
    x-maxDisplacement: 90
    x-spaceAround: 2
```

Add the following vendor option to the `point` define:

```
define: &point
  size: 6
  symbols:
  - mark:
      shape: circle
      stroke-color: 'black'
      stroke-width: 1
      fill-color: 'gray'
  x-labelObstacle: true
```



6. Now that we have clearly labeled our cities, zoom into an area you are familiar with and we can look at changing symbology on a case-by-case basis.

   We have used expressions previous to generate an appropriate label. Expressions can also be used for many other property settings.

   The `ne:populated_places` layer provides several attributes specifically to make styling easier:

- **SCALERANK**: we have already used this attribute to control the level of detail displayed
- **LABELRANK**: hint used for conflict resolution, allowing important cities such as capitals to be labeled even when they are close to a larger neighbor.
- **FEATURECLA**: used to indicate different types of cities. We will check for `Admin-0 capital` cities.

  The first thing we will do is calculate the point **size** using a quick expression:

```
${10-(SCALERANK/2)}
```

This expression should result in sizes between 5 and 9 and will need to be applied to both point **size** and label **displacement**.

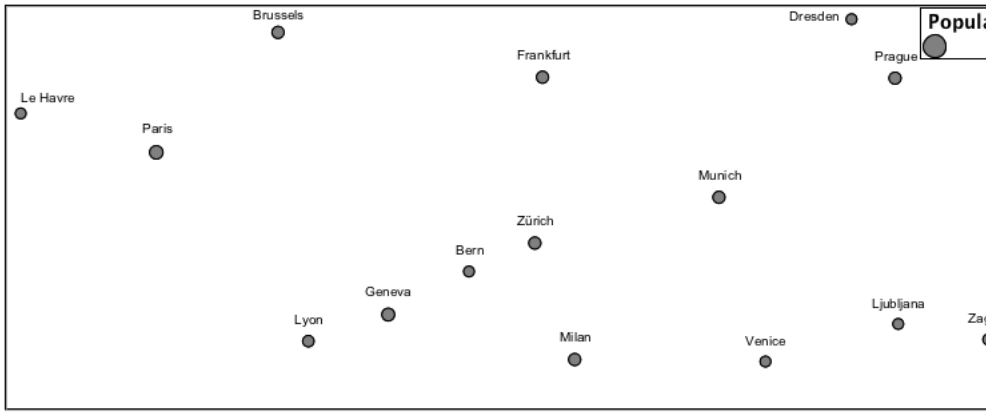
Rather than the "first come first served" default to resolve labeling conflicts we can manually provide GeoServer with a label priority. The expression provided is calculated for each label, in the event of a conflict the label with the highest priority takes precedence.

The LABELRANK attribute goes from 1 through 10 and needs to be flipped around before use as a GeoServer label priority:

```
${10 - LABELRANK}
```

This expression will result in values between 0 and 10 and will be used for the **priority**.

```
define: &point
  size: ${10-(SCALERANK/2)}
  symbols:
  - mark:
      shape: circle
      stroke-color: 'black'
      stroke-width: 1
      fill-color: 'gray'
  x-labelObstacle: true
define: &label
  label: ${NAME}
  fill-color: 'black'
  font-family: Arial
  font-size: 10
  font-style: normal
  font-weight: normal
  placement: point
  anchor: [0.5, 0]
  displacement: [0, '${''10'' - SCALERANK / ''2''}']
  priority: ${'10' - LABELRANK}
  x-maxDisplacement: 90
  x-spaceAround: 2
```



7. Next we can use `FEATURECLA` to check for capital cities.

   Adding a filter for capital cities at the top of the **rules** list:

```
- filter:␣
↪${SCALERANK < '2' AND FEATURECLA = 'Admin-0 capital'}
  scale: ['7.0E7', max]
  name: capitals
```

```
    symbolizers:
    - point:
        symbols:
        - mark:
            shape: star
            stroke-color: 'black'
            stroke-width: 1
            fill-color: 'gray'
    - text:
        label: ${NAME}
        fill-color: 'gray'
        placement: point
  - filter: ${FEATURECLA = 'Admin-0 capital'}
    scale: [min, '7.0E7']
    name: capitals
    symbolizers:
    - point:
        symbols:
        - mark:
            shape: star
            stroke-color: 'black'
            stroke-width: 1
            fill-color: 'gray'
    - text:
        label: ${NAME}
        fill-color: 'gray'
        placement: point
```

And updating the populated places filters to ignore capital cities:

```
  - filter: $
  ↪{SCALERANK < '7' AND FEATURECLA <> 'Admin-0 capital'}
    scale: ['4000000.0', '8000000.0']
    symbolizers:
    - point:
        <<: *point
    - text:
        <<: *label
  - filter: $
  ↪{SCALERANK < '5' AND FEATURECLA <> 'Admin-0 capital'}
    scale: ['8000000.0', '1.7E7']
    symbolizers:
    - point:
        <<: *point
    - text:
        <<: *label
  - filter: $
  ↪{SCALERANK < '4' AND FEATURECLA <> 'Admin-0 capital'}
    scale: ['1.7E7', '3.5E7']
    symbolizers:
    - point:
        <<: *point
    - text:
        <<: *label
  - filter: $
  ↪{SCALERANK < '3' AND FEATURECLA <> 'Admin-0 capital'}
    scale: ['3.5E7', '7.0E7']
    symbolizers:
```

```
      - point:
          <<: *point
      - text:
          <<: *label
    - filter: $
    ↪{SCALERANK < '2' AND FEATURECLA <> 'Admin-0 capital'}
      scale: ['7.0E7', '1.4E8']
      symbolizers:
      - point:
          <<: *point
      - text:
          <<: *label
    - filter: $
    ↪{SCALERANK < '1' AND FEATURECLA <> 'Admin-0 capital'}
      scale: ['1.4E8', max]
      symbolizers:
      - point:
          <<: *point
      - text:
          <<: *label
    - scale: [min, '4000000.0']
      symbolizers:
      - point:
          <<: *point
      - text:
          <<: *label
```

8. If you would like to check your work the final file is here: `point_example.ysld`

Bonus

Challenge Geometry Location

1. The **mark** property can be used to render any geometry content.
2. **Challenge:** Try this yourself by rendering a polygon layer using a **mark** property.

Note: Answer *discussed* at the end of the workbook.

Explore Dynamic Symbolization

1. We went to a lot of work to set up filters to choose between star and circle for capital cities.

   This approach is straightforward when applied in isolation:

```
rules:
- filter: ${FEATURECLA = 'Admin-0 capital'}
  scale: [min, max]
  symbolizers:
  - point:
      symbols:
      - mark:
          shape: star
          stroke-color: 'black'
          stroke-width: 1
          fill-color: 'gray'
- filter: ${FEATURECLA <> 'Admin-0 capital'}
  scale: [min, max]
  symbolizers:
  - point:
      symbols:
      - mark:
          shape: circle
          stroke-color: 'black'
          stroke-width: 1
          fill-color: 'gray'
```

   When combined with checking another attribute, or checking @scale as in our example, this approach can quickly lead to many rules which can be difficult to keep straight.

2. Taking a closer look, `shape` can actually be expressed using a string:

```
rules:
- filter: ${FEATURECLA = 'Admin-0 capital'}
  scale: [min, max]
  symbolizers:
  - point:
      symbols:
      - mark:
          shape: 'star'
          stroke-color: 'black'
          stroke-width: 1
          fill-color: 'gray'
```

   Which is represented in SLD as:

```
<sld:PointSymbolizer>
  <sld:Graphic>
    <sld:Mark>
      <sld:WellKnownName>star</sld:WellKnownName>
      <sld:Fill/>
      <sld:Stroke/>
    </sld:Mark>
  </sld:Graphic>
</sld:PointSymbolizer>
```

3. GeoServer recognizes this limitation of SLD Mark and ExternalGraphic and provides an opportunity for dynamic symbolization.

   This is accomplished by embedding a small CQL expression in the string passed to symbol or url. This sub-expression is isolated with `${ }` as shown:

```
- point:
    symbols:
    - mark:
        shape: ${if_then_else(equalTo(FEATURECLA,
↪'Admin-0 capital'),'star','circle')}
```

   Which is represented in SLD as:

```
<sld:PointSymbolizer>
  <sld:Graphic>
    <sld:Mark>
      <sld:WellKnownName>
↪${if_then_else(equalTo(FEATURECLA,'Admin-
↪0 capital'),'star','circle')}</sld:WellKnownName>
      <sld:Fill/>
      <sld:Stroke/>
    </sld:Mark>
  </sld:Graphic>
</sld:PointSymbolizer>
```

4. **Challenge:** Use this approach to rewrite the *Dynamic Styling* example.

Note: Answer *provided* at the end of the workbook.

Challenge Layer Group

1. Use a **Layer Group** to explore how symbology works together to form a map.

- ne:NE1
- ne:states_provincces_shp
- ne:populated_places

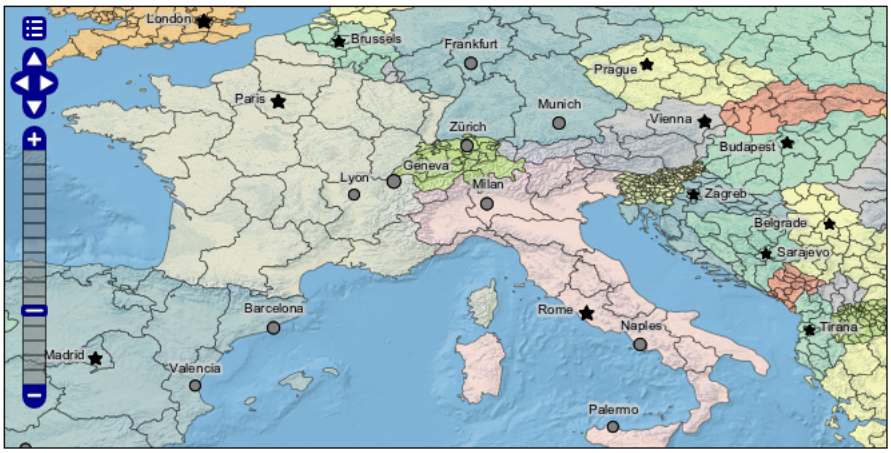
2. To help start things out here is a style for `ne:states_provinces_shp`:

```
symbolizers:
- polygon:
    stroke-color: 'black'
    stroke-width: 0.25
    stroke-opacity: 0.5
    fill-color: 'white'
    fill-opacity: 0.05
- polygon:
    stroke-color: 'black'
    stroke-width: 0.25
    stroke-opacity: 0.5
```

```
                                    fill-
                                ↪color: ${Recode(mapcolor9,'1','#8dd3c7','2','#ffffb3
                                ↪','3','#bebada','4','#fb8072','5','#80b1d3','6',
                                ↪'#fdb462','7','#b3de69','8','#fccde5','9','#d9d9d9')}
                                    fill-opacity: 0.5
```

3. This background is relatively busy and care must be taken to ensure both symbols and labels are clearly visible.
4. **Challenge:** Do your best to style populated_places over this busy background.

   Here is an example with labels for inspiration:



Note: Answer *provided* at the end of the workbook.

Explore True Type Fonts

1. In addition to image formats GeoServer can make use other kinds of graphics, such as True Type fonts:

```
symbolizers:
- point:
    symbols:
    - mark:
        shape: ttf://Webdings#0x0064
        stroke-color: 'blue'
        stroke-width: 1
```

2. Additional fonts dropped in the `styles` directory are available for use.

Explore Custom Graphics

1. The GeoServer rendering engine allows Java developers to hook in additional symbol support.

This facility is used by GeoServer to offer the shapes used for pattern fills. Community extensions allow the use of simple custom shapes and even charts.

2. Support has been added for custom graphics using the WKT Geometry representation.

```
symbolizers:
- point:
    symbols:
    - mark:
        shape: wkt://MULTILINESTRING((-
↪0.25 -0.25, -0.125 -0.25), (0.125 -0.25, 0.25 -0.25),
↪ (-0.25 0.25, -0.125 0.25), (0.125 0.25, 0.25 0.25))
        stroke-color: 'blue'
        stroke-width: 1
```

Rasters

Finally we will look at using YSLD styling for the portrayal of raster data.

Fig. 6.307: Raster Symbology

Review of raster symbology:

- Raster data is **Grid Coverage** where values have been recorded in a regular array. In OGC terms a **Coverage** can be used to look up a value or measurement for each location.
- When queried with a "sample" location:
  - A grid coverage can determine the appropriate array location and retrieve a value. Different techniques may be used interpolate an appropriate value from several measurements (higher quality) or directly return the "nearest neighbor" (faster).
  - A vector coverages would use a point-in-polygon check and return an appropriate attribute value.
  - A scientific model can calculate a value for each sample location
- Many raster formats organize information into bands of content. Values recorded in these bands and may be mapped into colors for display (a process similar to theming an attribute for vector data).

  For imagery the raster data is already formed into red, green and blue bands for display.
- As raster data has no inherent shape, the format is responsible for describing the orientation and location of the grid used to record measurements.

  These raster examples use a digital elevation model consisting of a single band of height measurements. The imagery examples use an RGB image that has been hand coloured for use as a base map.

  Reference:

- *YSLD Reference*
- *Raster* (YSLD Reference | Raster symbolizer)
- *Raster* (User Manual | SLD Reference)

The exercise makes use of the `usgs:dem` and `ne:ne1` layers.

Image

The **raster** symbolizer controls the display of raster data. By default, the raster symbolizer automatically selects the appropriate red, green and blue channels for display.

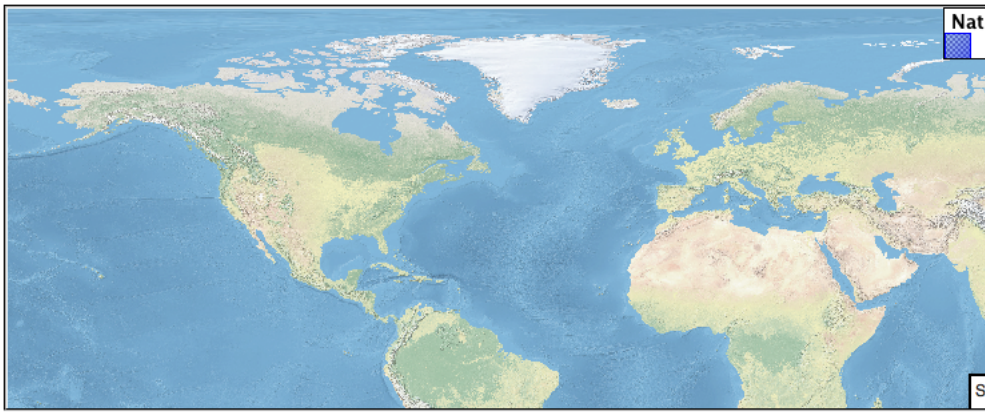
1. Navigate to the **Styles** page.
2. Click *Add a new style* and choose the following:

Name:	`image_example`
Workspace:	`No workspace`
Format:	`YSLD`

3. Choose *raster* from the `Generate a default style` dropdown and click *generate*.
4. Replace the initial YSLD definition with:

```
symbolizers:
- raster:
    opacity: 1.0
```

5. And use the *Layer Preview* tab to preview the result.



6. The **channels** property can be used to provide a list three band numbers (for images recording in several wave lengths) or a single band number can be used to view a grayscale image.

```
symbolizers:
- raster:
    opacity: 1.0
    channels:
      gray:
        name: '2'
```

7. Isolating just the green band (it wil be drawn as a grayscale image):

DEM

A digital elevation model is an example of raster data made up of measurements, rather than color information.

The `usgs:dem` layer used used for this exercise:

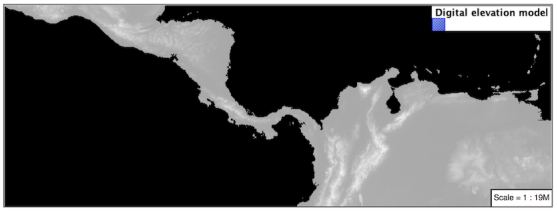
1. Return to the the **Styles** page.
2. Click *Add a new style* and choose the following:

Name:	`raster_example`
Workspace:	`No workspace`
Format:	`YSLD`

3. Choose *raster* from the `Generate a default style` dropdown and click *generate*.
4. The rendering engine will select our single band of raster content, and do its best to map these values into a grayscale image. Replace the content of the style with:

```
symbolizers:
- raster:
    opacity: 1.0
```

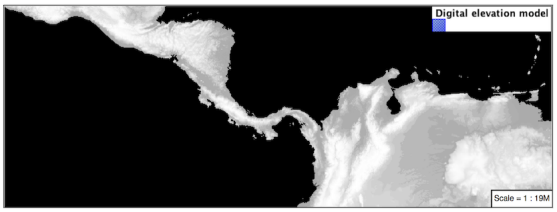
5. Use the *Layer Preview* tab to preview the result. The range produced in this case from the highest and lowest values.



6. We can use a bit of image processing to emphasis the generated color mapping by making use of **contrast-enhancement**.

```
symbolizers:
- raster:
    opacity: 1.0
    channels:
      gray:
        name: '1'
        contrast-enhancement:
          mode: histogram
```

7. Image processing of this sort should be used with caution as it does distort the presentation (in this case making the landscape look more varied then it is in reality.



Color Map

The approach of mapping a data channel directly to a color channel is only suitable to quickly look at quantitative data.

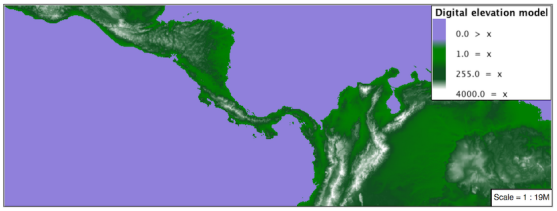
For qualitative data (such as land use) or simply to use color, we need a different approach:

Note: We can use a color map to artificially color a single band raster introducing smooth graduations for elevation or tempurature models or clear differentiation for qualitative data.

1. Apply the following YAML to our *usgs:DEM* layer:

```
symbolizers:
- raster:
    opacity: 1.0
    color-map:
      type: ramp
      entries:
      - ['#9080DB', 1.0, 0, null]
      - ['#008000', 1.0, 1, null]
      - ['#105020', 1.0, 255, null]
      - ['#FFFFFF', 1.0, 4000, null]
```

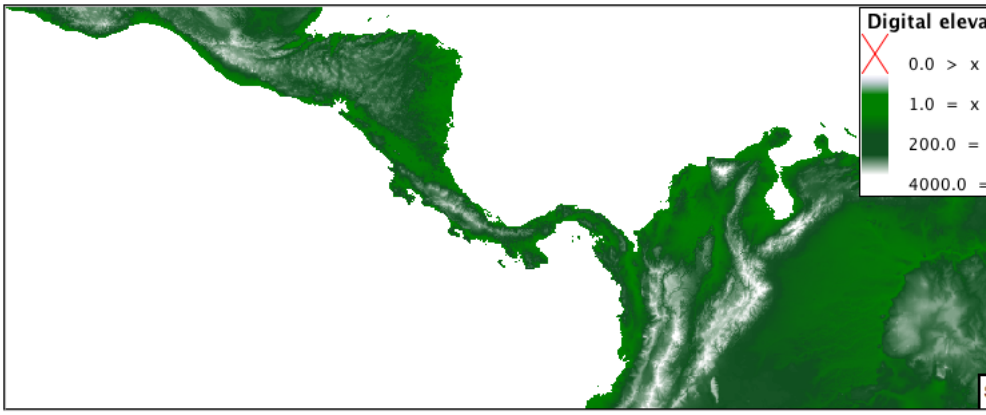
2. Resulting in this artificial color image:



3. An opacity value can also be used with each **color-map** entry.

```
symbolizers:
- raster:
    opacity: 1.0
    color-map:
      type: ramp
      entries:
      - ['#9080DB', 0.0, 0, null]
      - ['#008000', 1.0, 1, null]
      - ['#105020', 1.0, 255, null]
      - ['#FFFFFF', 1.0, 4000, null]
```

4. Allowing the areas of zero height to be transparent:



Note: Raster format for GIS work often supply a "no data" value, or contain a mask, limiting the dataset to only the locations with valid information.

Custom

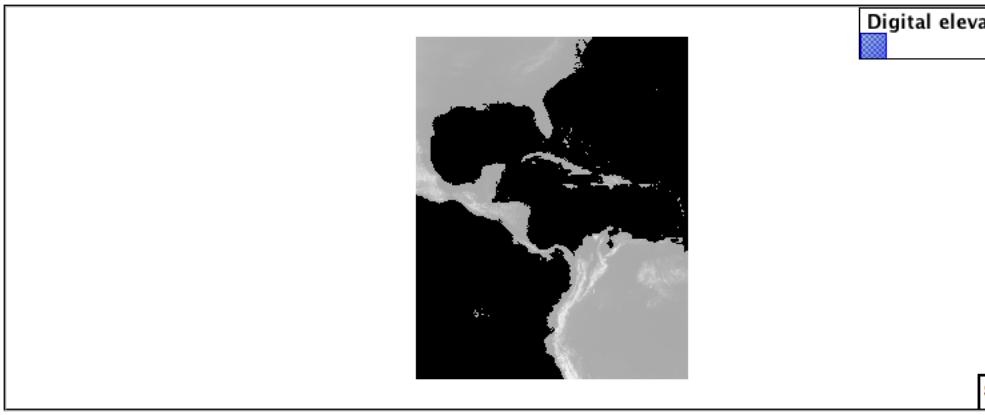
We can use what we have learned about color maps to apply a color brewer palette to our data.

This exploration focuses on accurately communicating differences in value, rather than strictly making a pretty picture. Care should be taken to consider the target audience and medium used during palette selection.

1. Restore the `raster_example` YSLD style to the following:

```
symbolizers:
- raster:
    opacity: 1.0
```

2. Producing the following map preview.

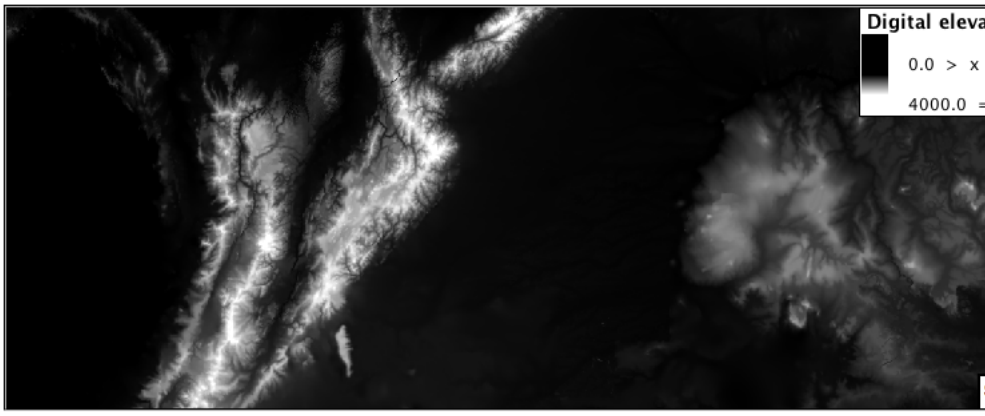


3. To start with we can provide our own grayscale using two color map entries.

```
symbolizers:
- raster:
    opacity: 1.0
    color-map:
      type: ramp
      entries:
      - ['#000000', 1.0, 0, null]
      - ['#FFFFFF', 1.0, 4000, null]
```

4. Use the *Layer Preview* tab to zoom in and take a look.

   This is much more direct representation of the source data. We have used our knowledge of elevations to construct a more accurate style.

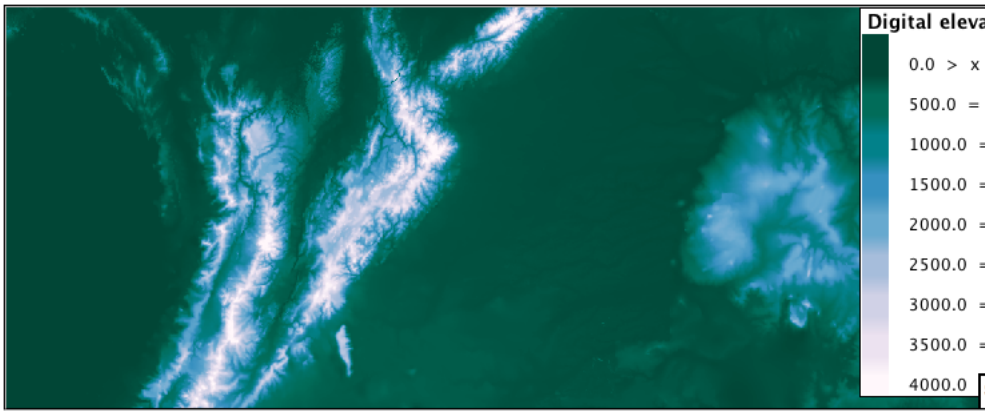


5. While our straightforward style is easy to understand, it does leave a bit to be desired with respect to clarity.

   The eye has a hard time telling apart dark shades of black (or bright shades of white) and will struggle to make sense of this image. To address this limitation we are going to switch to the ColorBrewer **9-class PuBuGn** palette. This is a sequential palette that has been hand tuned to communicate a steady change of values.

6. Update your style with the following:

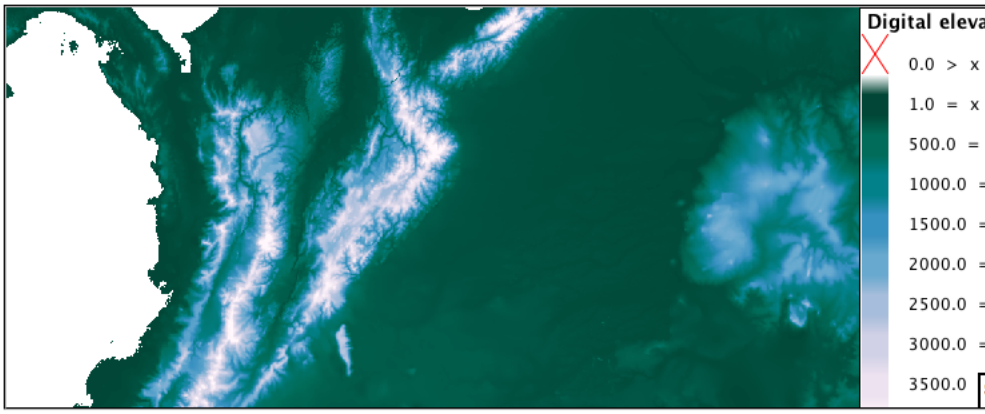
```
symbolizers:
- raster:
    opacity: 1.0
    color-map:
      type: ramp
      entries:
      - ['#014636', 1.0, 0, null]
      - ['#016C59', 1.0, 500, null]
      - ['#02818A', 1.0, 1000, null]
      - ['#3690C0', 1.0, 1500, null]
      - ['#67A9CF', 1.0, 2000, null]
      - ['#A6BDDB', 1.0, 2500, null]
      - ['#D0D1E6', 1.0, 3000, null]
      - ['#ECE2F0', 1.0, 3500, null]
      - ['#FFF7FB', 1.0, 4000, null]
```



7. A little bit of work with alpha (to mark the ocean as a no-data section):

```
symbolizers:
- raster:
    opacity: 1.0
    color-map:
      type: ramp
      entries:
      - ['#014636', 0, 0, null]
      - ['#014636', 1.0, 1, null]
      - ['#016C59', 1.0, 500, null]
      - ['#02818A', 1.0, 1000, null]
      - ['#3690C0', 1.0, 1500, null]
      - ['#67A9CF', 1.0, 2000, null]
      - ['#A6BDDB', 1.0, 2500, null]
      - ['#D0D1E6', 1.0, 3000, null]
      - ['#ECE2F0', 1.0, 3500, null]
      - ['#FFF7FB', 1.0, 4000, null]
```

8. And we are done:



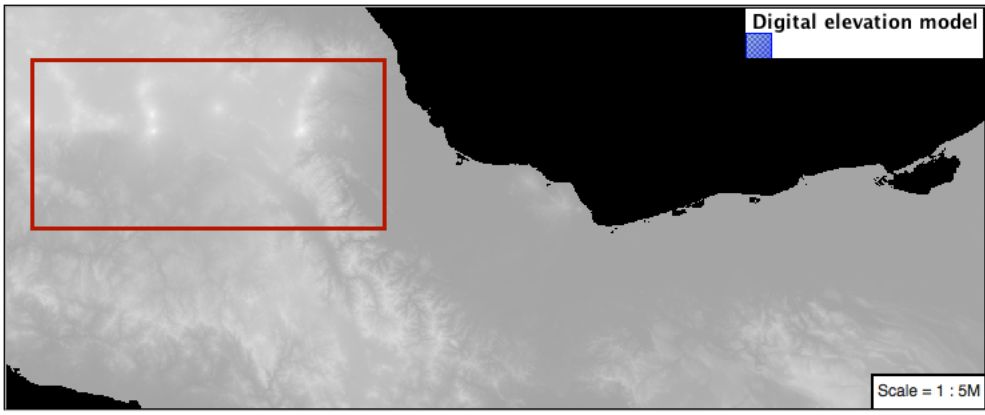
Bonus

Explore Contrast Enhancement

1. A special effect that is effective with grayscale information is automatic contrast adjustment.
2. Make use of a simple contrast enhancement with `usgs:dem`:

```
symbolizers:
- raster:
    opacity: 1.0
    contrast-enhancement:
      mode: normalize
```

1. Can you explain what happens when zoom in to only show a land area (as indicated with the bounding box below)?



Note: Discussion *provided* at the end of the workbook.

Challenge Intervals

1. The color-map **type** property dictates how the values are used to generate a resulting color.

- `ramp` is used for quantitative data, providing a smooth interpolation between the provided color values.
- `intervals` provides categorization for quantitative data, assigning each range of values a solid color.
- `values` is used for qualitative data, each value is required to have a **color-map** entry or it will not be displayed.

2. **Chalenge:** Update your DEM example to use **intervals** for presentation. What are the advantages of using this approach for elevation data?

Note: Answer *provided* at the end of the workbook.

Explore Image Processing

Additional properties are available to provide slight image processing during visualization.

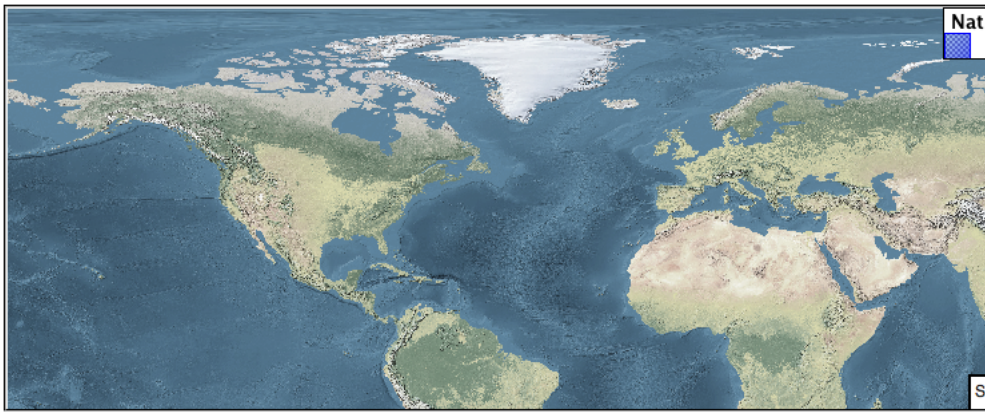
Note: In this section are we going to be working around a preview issue where only the top left corner of the raster remains visible during image processing. This issue has been reported as GEOS-6213.

Image processing can be used to enhance the output to highlight small details or to balance images from different sensors allowing them to be compared.

1. The **contrast-enhancement** property is used to turn on a range of post processing effects. Settings are provided for `normalize` or `histogram` or `none`;

```
symbolizers:
- raster:
    opacity: 1.0
    contrast-enhancement:
      mode: normalize
```

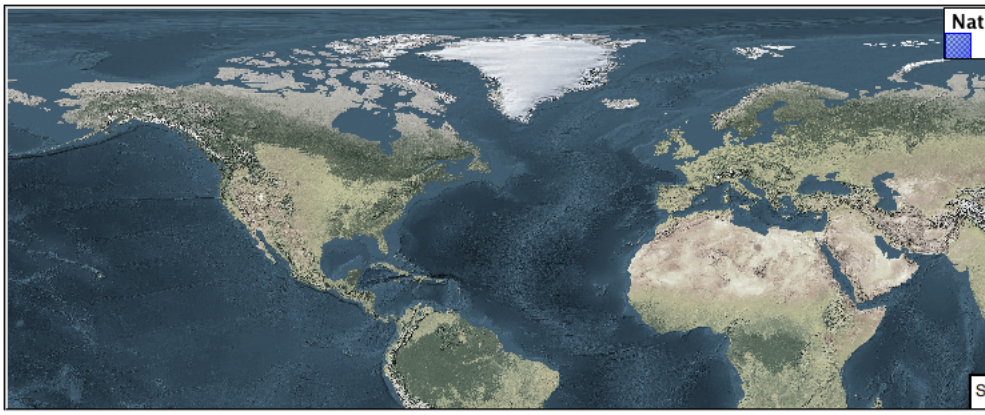
1. Producing the following image:



2. The **raster-gamma** property is used adjust the brightness of **contrast-enhancement** output. Values less than 1 are used to brighten the image while values greater than 1 darken the image.

```
symbolizers:
- raster:
    opacity: 1.0
    contrast-enhancement:
      gamma: 1.5
```

1. Providing the following effect:



Challenge Clear Digital Elevation Model Presentation

1. Now that you have seen the data on screen and have a better understanding how would you modify our initial gray-scale example?
2. **Challenge:** Use what you have learned to present the `usgs:dem` clearly.

Note: Answer *provided* at the end of the workbook.

Challenge Raster Opacity

1. There is a quick way to make raster data transparent, raster **opacity** property works in the same fashion as with vector data. The raster as a whole will be drawn partially transparent allow content from other layers to provide context.
2. **Challenge:** Can you think of an example where this would be useful?

Note: Discussion *provided* at the end of the workbook.

YSLD Workbook Conclusion

We hope you have enjoyed this styling workshop.

Additional resources:

- *YSLD Extension*
- *YSLD Reference*

YSLD Tips and Tricks

Converting to YSLD

The REST API can be used to convert any of your existing CSS or SLD styles to YSLD.

1. Navigate to the rest api endpoint for styles:

- view-source:http://localhost:8080/geoserver/rest/styles

> **Note:** Using `view-source:` in chrome or firefox allows us to focus on page content.

2. Click on one of the styles (for example `states.html`)
3. Click on the link to the style contents

- view-source:http://localhost:8080/geoserver/rest/styles/states.sld

4. Change the URL with `?pretty=true` for human readable XML.

- view-source:http://localhost:8080/geoserver/rest/styles/states.sld?pretty=true

5. Change the URL with `yaml` to convert to YSLD.

- view-source:http://localhost:8080/geoserver/rest/styles/states.yaml

The original SLD file is convert to YSLD:

```
name: states
title: Population in the United States
abstract: |-
  A sample␣
↪filter that filters the United States into three
      ␣
↪  categories of population, drawn in different colors
feature-styles:
- name: name
  rules:
  - name: Population < 2M
    title: Population < 2M
    filter: ${PERSONS < '2000000'}
    scale: [min, max]
    symbolizers:
    - polygon:
        fill-color: '#A6CEE3'
        fill-opacity: 0.7
  - name: Population 2M-4M
```

```
    title: Population 2M-4M
    filter: ${PERSONS BETWEEN '2000000' AND '4000000'}
    scale: [min, max]
    symbolizers:
    - polygon:
        fill-color: F78B4
        fill-opacity: 0.7
  - name: '> 4M'
    title: Population > 4M
    filter: ${PERSONS > '4000000'}
    scale: [min, max]
    symbolizers:
    - polygon:
        fill-color: '#B2DF8A'
        fill-opacity: 0.7
  - name: State Outlines
    title: State Outlines
    scale: [min, max]
    symbolizers:
    - line:
        stroke-color: '#8CADBF'
        stroke-width: 0.1
  - name: State Abbreviations
    title: State Abbreviations
    scale: ['1.75E7', '3.5E7']
    symbolizers:
    - text:
        label: ${STATE_ABBR}
        font-family: SansSerif
        font-size: 12
        font-style: Normal
        font-weight: normal
        placement: point
        anchor: [0.5, 0.5]
  - name: State Names
    title: State Names
    scale: [min, '1.75E7']
    symbolizers:
    - text:
        label: ${STATE_NAME}
        font-family: SansSerif
        font-size: 12
        font-style: Normal
        font-weight: normal
        placement: point
        anchor: [0.5, 0.5]
        x-maxDisplacement: 100
        x-goodnessOfFit: 0.9
```

YSLD Workshop Answer Key

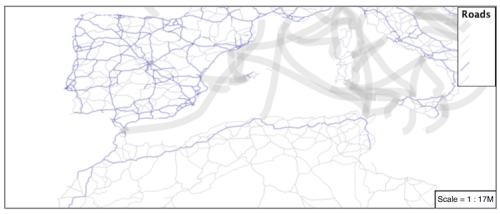
The following questions were listed through out the workshop as an opportunity to explore the material in greater depth. Please do your best to consider the questions in detail prior to checking here for the answer. Questions are provided to teach valuable skills, such as a chance to understand how feature type styles are used to control

z-order, or where to locate information in the user manual.

Classification

Answer for *Challenge Classification*:

1. **Challenge:** Create a new style adjust road appearance based on **type**.



   Hint: The available values are 'Major Highway','Secondary Highway','Road' and 'Unknown'.

2. Here is an example:

```
define: &common
  stroke-opacity: 0.25

rules:
- filter: ${type = 'Major Highway'}
  symbolizers:
  - line:
      stroke-color: '#000088'
      stroke-width: 1.25
      <<: *common
- filter: ${type = 'Secondary Highway'}
  symbolizers:
  - line:
      stroke-color: '#8888AA'
      stroke-width: 0.75
      <<: *common
- filter: ${type = 'Road'}
  symbolizers:
  - line:
      stroke-color: '#888888'
      stroke-width: 0.75
      <<: *common
- filter: ${type = 'Unknown'}
  symbolizers:
  - line:
      stroke-color: '#888888'
      stroke-width: 0.5
      <<: *common
- else: true
  symbolizers:
  - line:
      stroke-color: '#AAAAAA'
      stroke-width: 0.5
      <<: *common
```

One Rule Classification

Answer for *Challenge One Rule Classification*:

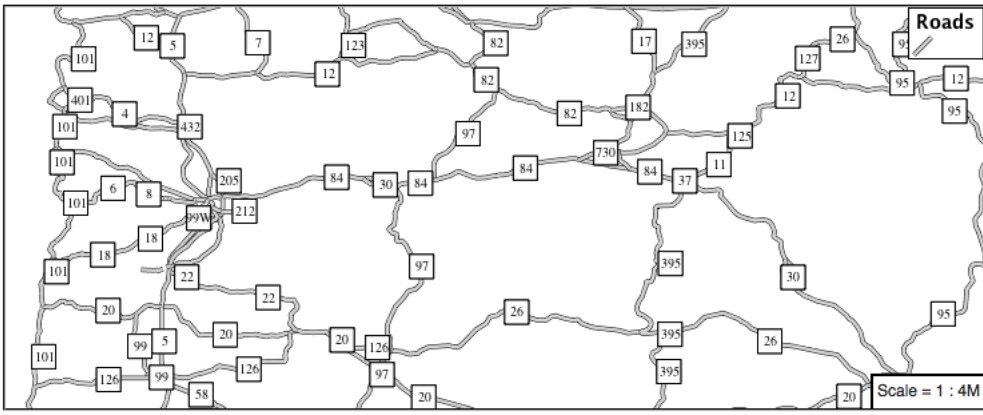
1. **Challenge:** Create a new style and classify the roads based on their scale rank using expressions in a single rule instead of multiple rules with filters.
2. This exercise requires looking up information in the user guide, the search tearm *recode* provides several examples.

- The YSLD Reference *theming functions* provides a clear example.

Label Shields

Answer for *Challenge Label Shields*:

1. *Challenge:* Have a look at the documentation for putting a graphic on a text symbolizer in SLD and reproduce this technique in YSLD.



2. The use of a label shield is a vendor specific capability of the GeoServer rendering engine. The tricky part of this exercise is finding the documentation online (i.e. *TextSymbolizer - Graphic*).

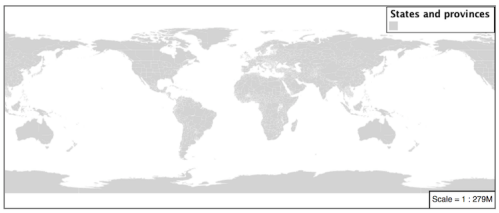
```
symbolizers:
- line:
    stroke-color: '#000000'
    stroke-width: 3
- line:
    stroke-color: '#D3D3D3'
    stroke-width: 2
- text:
    label: ${name}
    fill-color: '#000000'
    font-family: Ariel
    font-size: 10
    font-style: normal
    font-weight: normal
    placement: point
    graphic:
      size: 18
      symbols:
      - mark:
```

```
                                    shape: square
                                    stroke-color: '#000000'
                                    stroke-width: 1
                                    fill-color: '#FFFFFF'
```

Antialiasing

Answer for *Explore Antialiasing*:

1. When we rendered our initial preview, without a stroke, thin white gaps (or slivers) are visible between our polygons.



   This effect is made more pronounced by the rendering engine making use of the Java 2D sub-pixel accuracy. This technique is primarily used to prevent an aliased (stair-stepped) appearance on diagonal lines.

2. **Explore:** Experiment with **fill** and **stroke** settings to eliminate slivers between polygons.

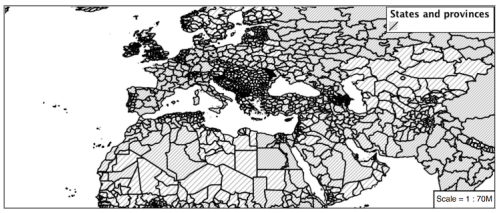
   The obvious approach works - setting both values to the same color:

```
                                symbolizers:
                                - polygon:
                                    stroke-color: 'lightgrey'
                                    stroke-width: 1
                                    fill-color: 'lightgrey'
```

Categorize

Answer for *Explore Categorize*:

1. An exciting use of the GeoServer **shape** symbols is the theming by changing the **size** used for pattern density.
2. **Explore:** Use the **Categorize** function to theme by **datarank**.



   Example:

```
symbolizers:
- polygon:
    stroke-color: 'black'
    stroke-width: 1
    fill-color: 'gray'
    fill-graphic:
      size:␣
↪${Categorize(datarank,'4','4','5','6','8','10','10')}
      symbols:
      - mark:
          shape: shape://slash
          stroke-color: 'darkgray'
          stroke-width: 1
```

Halo

Answer for *Challenge Halo*:

1. The halo example used the fill color and opacity for a muted halo, while this improved readability it did not bring attention to our labels.

   A common design choice for emphasis is to outline the text in a contrasting color.

2. **Challenge:** Produce a map that uses a white halo around black text.

   Here is an example:

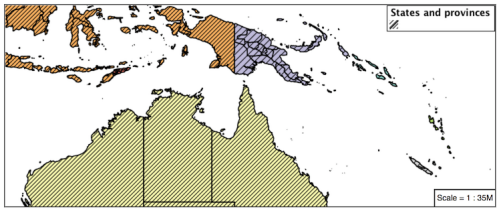
```
symbolizers:
- polygon:
    stroke-color: 'gray'
    stroke-width: 1
    fill-color: '#7EB5D3'
- text:
    label: ${name}
    fill-color: 'black'
    halo:
      fill-color: 'white'
      radius: 1
    font-family: Arial
    font-size: 14
    font-style: normal
    font-weight: normal
    anchor: [0.5, 0.5]
```

Theming using Multiple Attributes

Answer for *Challenge Theming using Multiple Attributes*:

1. A powerful tool is theming using multiple attributes. This is an important concept allowing map readers to perform "integration by eyeball" (detecting correlations between attribute values information).

2. **Challenge:** Combine the **mapcolor9** and **datarank** examples to reproduce the following map.



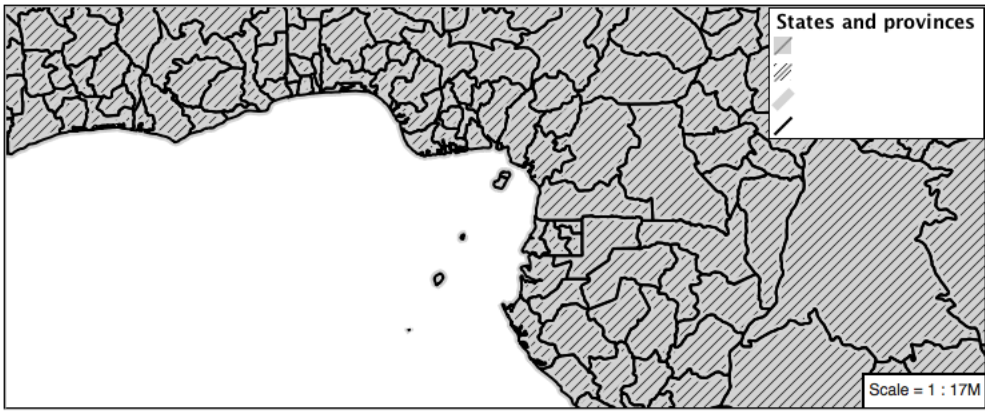
This should be a cut and paste using the `recode` example, and `categorize` examples already provided.

```
symbolizers:
- polygon:
    stroke-color: 'black'
    stroke-width: 1
    fill-color: ${Recode(mapcolor9,
      '1','#8dd3c7',
      '2','#ffffb3',
      '3','#bebada',
      '4','#fb8072',
      '5','#80b1d3',
      '6','#fdb462',
      '7','#b3de69',
      '8','#fccde5',
      '9','#d9d9d9')}
- polygon:
    stroke-color: 'black'
    stroke-width: 1
    fill-color: 'gray'
    fill-graphic:
      size: $
↪{Categorize(datarank,'6','4','8','6','10','10','12')}
      symbols:
      - mark:
          shape: shape://slash
          stroke-color: 'black'
          stroke-width: 1
          fill-color: 'gray'
```

Use of Feature styles

Answer for *Challenge Use of Feature styles*:

1. Using multiple **feature-styles** to simulate line string casing. The resulting effect is similar to text halos - providing breathing space around complex line work allowing it to stand out.
2. **Challenge:** Use what you know of LineString **feature-styles** to reproduce the following map:



This is much easier when using YSLD, where z-order is controlled by feature-style order. In this instance, multiple symbolizers within a feature-style will not work, as the order within a feature-style is only consistent per-feature (not per-layer).

```
feature-styles:
- rules:
  - symbolizers:
    - polygon:
        stroke-width: 1.0
        fill-color: 'lightgrey'
- rules:
  - symbolizers:
    - polygon:
        stroke-width: 1.0
        fill-color: 'gray'
        fill-graphic:
          size: 8
          symbols:
          - mark:
              shape: shape://slash
              stroke-color: 'black'
              stroke-width: 0.75
- rules:
  - symbolizers:
    - line:
        stroke-color: 'lightgrey'
        stroke-width: 6
- rules:
  - symbolizers:
    - line:
        stroke-color: 'black'
        stroke-width: 1.5
```

The structure of the legend graphic provides an indication on what is going on.

Geometry Location

Answer for *Challenge Geometry Location*:

1. The **mark** property can be used to render any geometry content.

2. **Challenge:** Try this yourself by rendering a polygon layer using a **mark** property.

   This can be done one of two ways:

- Changing the association of a polygon layer, such as `ne:states_provinces_shp` to point_example and using the layer preview page.
- Changing the *Layer Preview* tab to a polygon layer, such as `ne:states_provinces_shp`.

  The important thing to notice is that the centroid of each polygon is used as a point location.

Dynamic Symbolization

Answer for *Explore Dynamic Symbolization*:

1. SLD Mark and ExternalGraphic provide an opportunity for dynamic symbolization.

   This is accomplished by embedding a small CQL expression in the string passed to symbol or url. This sub-expression is isolated with `${ }` as shown:

```
- point:
    symbols:
    - mark:
        shape: ${if_then_else(equalTo(FEATURECLA,
↪'Admin-0 capital'),'star','circle')}
```

2. **Challenge:** Use this approach to rewrite the *Dynamic Styling* example.

   Example available here `point_example.css`:

Layer Group

Answer for *Challenge Layer Group*:

1. Use a **Layer Group** to explore how symbology works together to form a map.

- ne:NE1
- ne:states_provincces_shp
- ne:populated_places

2. This background is relatively busy and care must be taken to ensure both symbols and labels are clearly visible.
3. **Challenge:** Do your best to style populated_places over this busy background.

   Here is an example with labels for inspiration:

This is opportunity to revisit label halo settings from *Polygons*:

```
symbolizers:
- point:
    size: ${'5' + '10' - SCALERANK / '3'}
    symbols:
    - mark:
        shape: circle
        stroke-color: 'white'
        stroke-width: 1
        stroke-opacity: 0.75
        fill-color: 'black'
        x-labelObstacle: true
    - text:
        label: ${name}
        fill-color: 'black'
        font-family: Arial
        font-size: 14
        anchor: [0.5, 1]
        offset: [0 ${'-12' + SCALERANK}]
        halo:
          fill-color: `lightgray`
          radius: 2
          opacity: 0.7
        priority: ${`0` - LABELRANK}
        x-maxDisplacement: 90
```

Using a lightgray halo, 0.7 opacity and radius 2 fades out the complexity immediately surrounding the label text improving legibility.

Contrast Enhancement

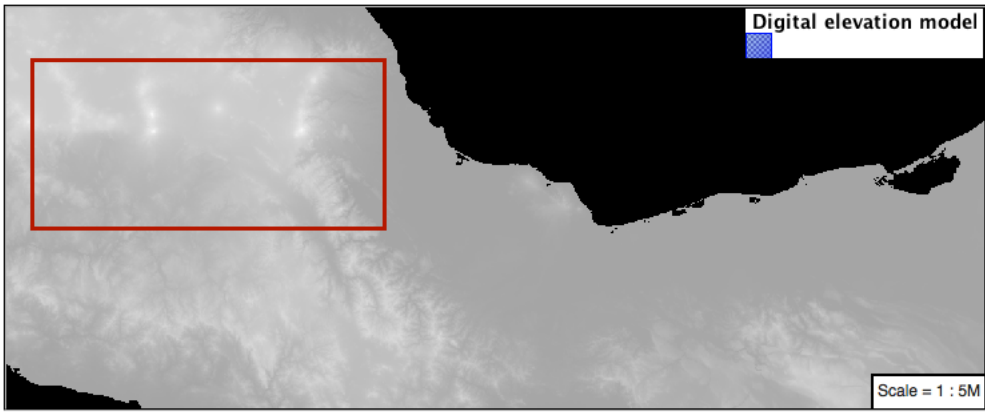
Discussion for *Explore Contrast Enhancement*:

1. A special effect that is effective with grayscale information is automatic contrast adjustment.
2. Make use of a simple contrast enhancement with `usgs:dem`:

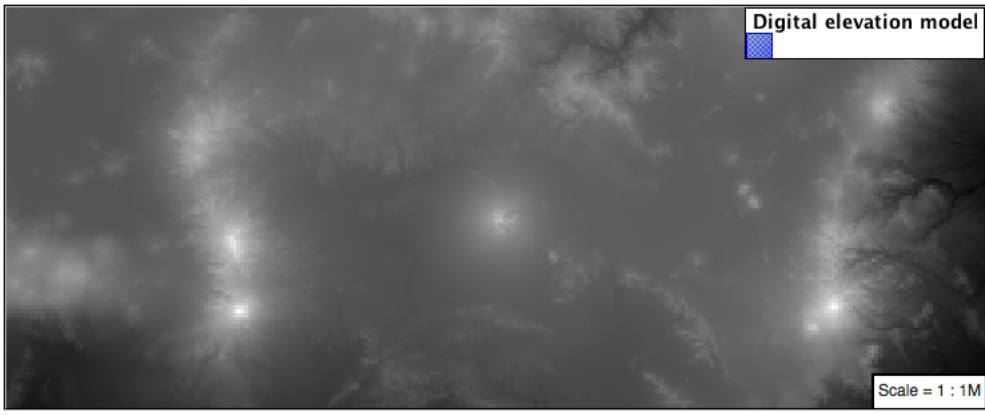
```
symbolizers:
- raster:
    opacity: 1.0
```

```
                                    contrast-enhancement:
                                      mode: normalize
```

3. Can you explain what happens when zoom in to only show a land area (as indicated with the bounding box below)?



What happens is insanity, normalize stretches the palette of the output image to use the full dynamic range. As long as we have ocean on the screen (with value 0) the land area was shown with roughly the same presentation.



Once we zoom in to show only a land area, the lowest point on the screen (say 100) becomes the new black, radically altering what is displayed on the screen.

Intervals

Answer for *Challenge Intervals*:

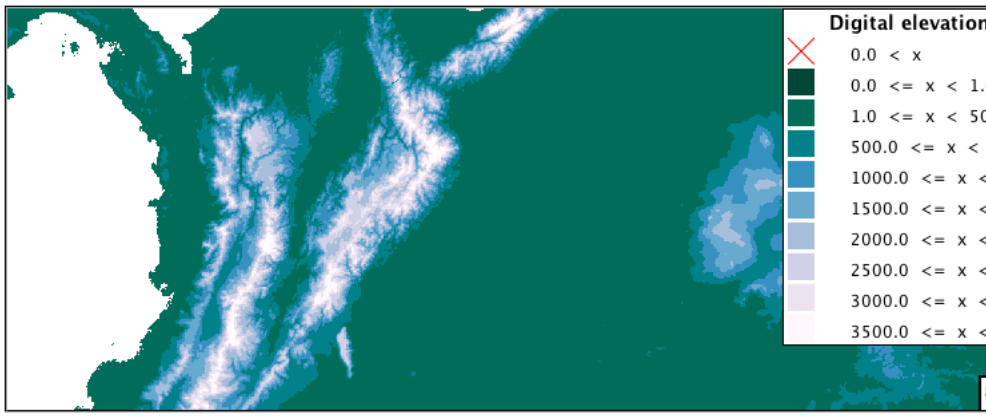
1. The color-map **type** property dictates how the values are used to generate a resulting color.

- `ramp` is used for quantitative data, providing a smooth interpolation between the provided color values.
- `intervals` provides categorization for quantitative data, assigning each range of values a solid color.

- `values` is used for qualitative data, each value is required to have a **color-map** entry or it will not be displayed.

2. **Chalenge:** Update your DEM example to use **intervals** for presentation. What are the advantages of using this approach for elevation data?

   By using intervals it becomes very clear how relatively flat most of the continent is. The ramp presentation provided lots of fascinating detail which distracted from this fact.



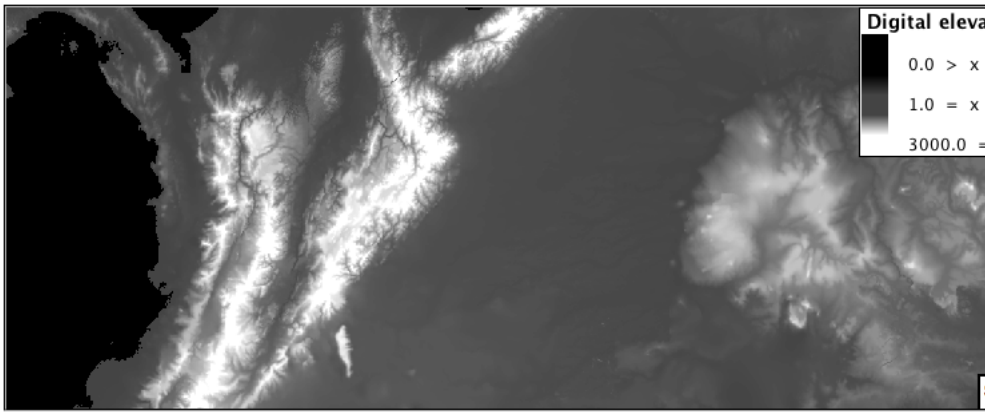
   Here is style for you to cut and paste:

```
symbolizers:
- raster:
    opacity: 1.0
    color-map:
      type: intervals
      entries:
      - ['#014636', 0, 0, null]
      - ['#014636', 1.0, 1, null]
      - ['#016C59', 1.0, 500, null]
      - ['#02818A', 1.0, 1000, null]
      - ['#3690C0', 1.0, 1500, null]
      - ['#67A9CF', 1.0, 2000, null]
      - ['#A6BDDB', 1.0, 2500, null]
      - ['#D0D1E6', 1.0, 3000, null]
      - ['#ECE2F0', 1.0, 3500, null]
      - ['#FFF7FB', 1.0, 4000, null]
```

Clear Digital Elevation Model Presentation

Answer for *Challenge Clear Digital Elevation Model Presentation*:

1. Now that you have seen the data on screen and have a better understanding how would you modify our initial gray-scale example?
2. **Challenge:** Use what you have learned to present the `usgs:dem` clearly.



The original was a dark mess. Consider making use of mid-tones (or adopting a sequential palette from color brewer) in order to fix this. In the following example the ocean has been left dark, allowing the mountains stand out more.

```
symbolizers:
- raster:
    opacity: 1.0
    color-map:
      type: ramp
      entries:
      - ['#000000', 1.0, 0, null]
      - ['#444444', 1.0, 1, null]
      - ['#FFFFFF', 1.0, 3000, null]
```

Raster Opacity

Discussion for *Challenge Clear Digital Elevation Model Presentation*:

1. There is a quick way to make raster data transparent, raster **opacity** property works in the same fashion as with vector data. The raster as a whole will be drawn partially transparent allow content from other layers to provide context.
2. **Challenge:** Can you think of an example where this would be useful?

   This is difficult as raster data is usually provided for use as a basemap, with layers being drawn over top.

   The most obvious example here is the display of weather systems, or model output such as fire danger. By drawing the raster with some transparency, the landmass can be shown for context.

6.7.5 MBStyle Styling Workbook

GeoServer styling can be used for a range of creative effects. This section introduces the *MBStyle Extension* which can be used as alternative to SLD files.

MBStyle Quickstart

In the last section, we saw how the OGC defines style using XML documents (called SLD files).

We will now explore GeoServer styling in greater detail using a tool to generate our SLD files. The **MBStyle** GeoServer extension is used to generate SLD files using the **MabBox Style** styling language. Styles written in this language can also be used to style *vector tiles* in client-side applications.

Using the MBStyle extension to define styles results in shorter examples that are easier to understand. At any point we will be able to review the generated SLD file.

Reference:

- *MBStyle Reference*

MBStyle Syntax

This section provides a quick introduction to MBStyle syntax for mapping professionals who may not be familiar with JSON.

JSON Syntax

All MBStyles consist of a JSON document. There are three types of structures in a JSON document:

1. Object, a collection of key-value pairs. All JSON documents are JSON objects.
2. Array, a collection of values.
3. Value, the value in a key-value pair, or an entry in an array. Values can be objects, arrays, strings, numbers, *true*, *false*, or *null*.

Object	A collection of key-value pairs, enclosed by curly braces and delimited by commas. Keys are surronded by quotes and seperarted from values by a colon.
Array	A collection values, enclosed by square brackets and delimited by commas.
String	Text value. Must be surrounded by quotes.
Number	Numerical value. Must not be surrounded by quotes.
Boolean	*true* or *false*.
Null	*null*. Represents an undefined or unset value.

MBStyle Specification

The Mapbox Style specification defines a number of additional rules that MBStyles must follow.

Root-level Properties

Root level properties of a Mapbox style specify the map's layers, tile sources and other resources, and default values for the initial camera position when not specified elsewhere.

The following root-level properties are required for all MBStyles. Additional root-level properties which are supported but not required can be found in the spec.

version	The version of the Mapbox Style specification to use. Must be set to *8*.
name	The name of the style.
sources	An object defining the source data. Not used by GeoServer.
layers	An array of layer style objects

For example:

```
{
    "version": 8,
    "name": "Streets",
    "sources": {...},
    "layers": [...]
}
```

Sources

The sources parameter consists of a collection of named sources which define vector tile data the style is to be applied to. This is only used for MBStyles used in client-side applications, and is ignored by GeoServer. If you are only using MBStyles to style your layers within GeoServer, you don't need a sources parameter. However, if you also want to use your MBStyles for client-side styling, you will need the sources parameter.

A GeoServer vector tile source would be defined like this:

```
{
  "cookbook": {
    "type": "vector",
    "tiles": [
      "http://localhost:8080/geoserver/gwc/
↪service/wmts?REQUEST=GetTile&SERVICE=WMTS&VERSION=1.
↪0.0&LAYER=cookbook&STYLE=&TILEMATRIX=EPSG:900913:
↪{z}&TILEMATRIXSET=EPSG:900913&FORMAT=application/x-
↪protobuf;type=mapbox-vector&TILECOL={x}&TILEROW={y}"
    ],
    "minZoom": 0,
    "maxZoom": 14
  }
}
```

Layers

The layers parameter contains the primary layout and styling information in the MBStyle. Each layer in the layers list is a self-contained block of styling information. Layers are applied in order, so the last layer in the layers list will be rendered at the top of the image.

Note: A layer in an MBStyle is not the same as a layer in GeoServer. A GeoServer layer is a raster or vector dataset that represents a collection of geographic features. A MBStyle layer is a block of styling information, similar to a SLD Symbolizer.

<Example>

Reference:

- *MBStyle Styling* (User Guide)
- Mapbox Style specification

Compare MBStyle to SLD

The MBStyle extension is built with the same GeoServer rendering engine in mind, providing access to most of the functionality of SLD. The two approaches use slightly different terminology: SLD uses terms familiar to mapping professionals, while MBStyle uses ideas more familiar to web developers.

SLD Style

Here is an example `SLD file` for reference:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<StyledLayerDescriptor version="1.0.0"
 xsi:schemaLocation="http:/
↪/www.opengis.net/sld StyledLayerDescriptor.xsd"
 xmlns="http://www.opengis.net/sld"
 xmlns:ogc="http://www.opengis.net/ogc"
 xmlns:xlink="http://www.w3.org/1999/xlink"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <NamedLayer>
    <Name>airports</Name>
    <UserStyle>
      <Title>Airports</Title>
      <FeatureTypeStyle>
        <Rule>
          <Name>airports</Name>
          <Title>Airports</Title>
          <PointSymbolizer>
            <Graphic>
              <ExternalGraphic>
                <OnlineResource xlink:type="simple"
                xlink:href="airport.svg" />
                <Format>image/svg</Format>
              </ExternalGraphic>
```

```
                                    <Size>16</Size>
                                  </Graphic>
                                </PointSymbolizer>
                              </Rule>
                            </FeatureTypeStyle>
                          </UserStyle>
                        </NamedLayer>
                      </StyledLayerDescriptor>
```

MBStyle Style

Here is the same example as `MBStyle`:

```
{
  "version": 8,
  "name": "airports",
  "sprite
↪": "http://localhost:8080/geoserver/styles/sprites",
  "layers": [
      {
          "id": "airports",
          "type": "symbol",
          "layout": {
              "icon-image": "airport"
          }
      }
  ]
}
```

We use a point symbolizer to indicate we want this content drawn as a **Point** (line 16 in the SLD, line 8 in the MBStyle). The point symbolizer declares an external graphic, which contains the URL `airports.svg` indicating the image that should be drawn (line 20 in the SLD, line 10 in the MBStyle).

Note: Rather than refer to many diffferent icons seperatly, MBStyles use a single spritesheet containing all the necessary icons for the style. This is defined by the `sprite` property at the top-level of the style.

Tour

To confirm everything works, let's reproduce the airports style above.

1. Navigate to the **Styles** page.
2. Each time we edit a style, the contents of the associated SLD file are replaced. Rather then disrupt any of our existing styles we will create a new style. Click *Add a new style* and choose the following:

Name:	`airports0`
Workspace:	(leave empty)
Format:	`MBStyle`

3. Replace the initial MBStyle definition with with our airport MBStyle example and click *Apply*:

```
{
  "version": 8,
  "name": "airports",
  "sprite
↪": "http://localhost:8080/geoserver/styles/sprites",
  "layers": [
      {
          "id": "airports",
          "type": "symbol",
          "layout": {
              "icon-image": "airport"
          }
      }
  ]
}
```

4. Click the *Layer Preview* tab to preview the style. We want to preview on the aiports layer, so click the name of the current layer and select `ne:airports` from the list that appears. You can use the mouse buttons to pan and scroll wheel to change scale.

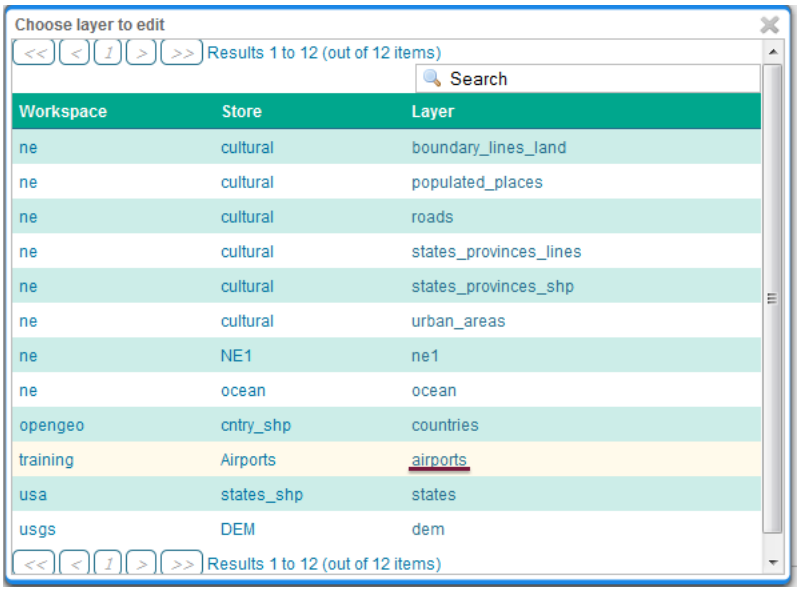


Fig. 6.308: Choosing the airports layer

5. Click *Layer Data* for a summary of the selected data.

Bonus

Finished early? For now please help your neighbour so we can proceed with the workshop.

If you are really stuck please consider the following challenge rather than skipping ahead.

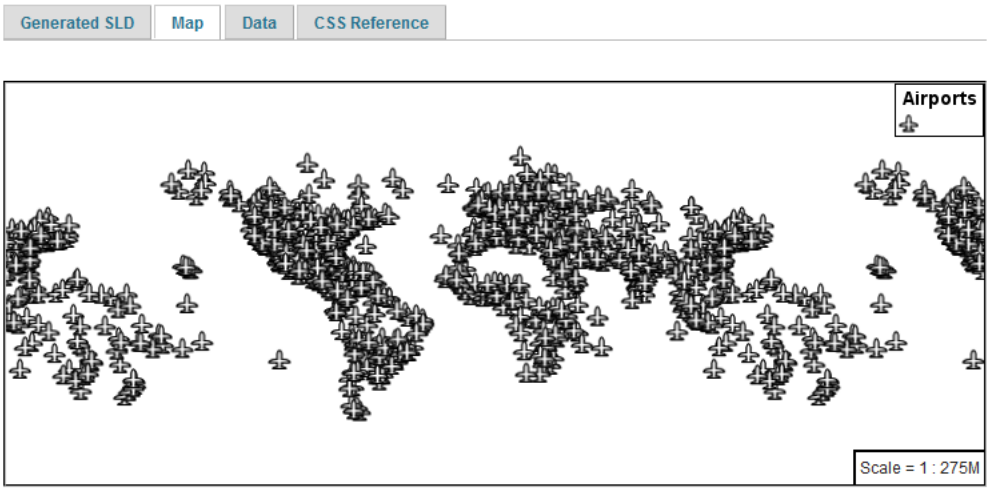


Fig. 6.309: Layer preview

Generated SLD | Map | Data | CSS Reference

For reference, here is a listing of the attributes in this data set.

Name	Type	Sample value	Min	Max	Compute stats
the_geom	Point	POINT (75.95707224036518 30.850359856170176)			Compute
scalerank	Integer	9	2	9	Compute
featurecla	String	Airport			Compute
type	String	small			Compute
name	String	Sahnewal			Compute
abbrev	String	LUH			Compute
location	String	terminal			Compute
gps_code	String	VILD			Compute
iata_code	String	LUH			Compute
wikipedia	String	http://en.wikipedia.org/wiki/Sahnewal_Airport			Compute
natlscale	Double	8.0			Compute

Fig. 6.310: Layer attributes

Explore Data

1. Return to the *Data* tab and use the *Compute* link to determine the minimum and maximum for the **scalerank** attribute.

Challenge Compare SLD Generation

1. The rest API can be used to review your YAML file directly.

   Browser:

- view-source:http://localhost:8080/geoserver/rest/styles/airport0.json

   Command line:

```
curl -v -u admin:geoserver -XGET http:/
↪/localhost:8080/geoserver/rest/styles/airports0.json
```

1. The REST API can also be used generate an SLD file:

   Browser:

- view-source:http://localhost:8080/geoserver/rest/styles/airport0.sld?pretty=true

   Command line:

```
curl -v -u admin:geoserver -XGET http://localhost:8080/
↪geoserver/rest/styles/airports0.sld?pretty=true
```

1. Compare the generated SLD differ above with the hand written `SLD file` used as an example?

   Challenge: What differences can you spot?

Lines

We will start our tour of MBStyle styling by looking at the representation of lines.

Fig. 6.311: LineString Geometry

Review of line symbology:

- Lines can be used to represent either abstract concepts with length but not width such as networks and boundaries, or long thin features with a didth that is too smallt o represent on the map. This means that **the visual width of line symbols do not normally change depending on scale.**
- Lines are recorded as LineStrings or Curves depending on the geometry model used.
- SLD uses a **LineSymbolizer** to record how the shape of a line is drawn. The primary characteristic documented is the **Stroke** used to draw each segment between vertices.

- Labeling of line work is anchored to the mid-point of the line. GeoServer provides a vendor option to allow label rotation aligned with line segments.

  For our exercises we are going to be using simple MBStyle documents, often consisting of a single layer, in order to focus on the properties used for line symbology.

  Each exercise makes use of the `ne:roads` layer.

  Reference:

- *MBStyle Reference*
- MapBox Style Spec Line Layer
- *LineString* (User Manual | SLD Reference)

Line

A line layer is represented by the `line` type.

Fig. 6.312: Basic Stroke Properties

1. Navigate to the **Styles** page.
2. Click *Add a new style* and choose the following:

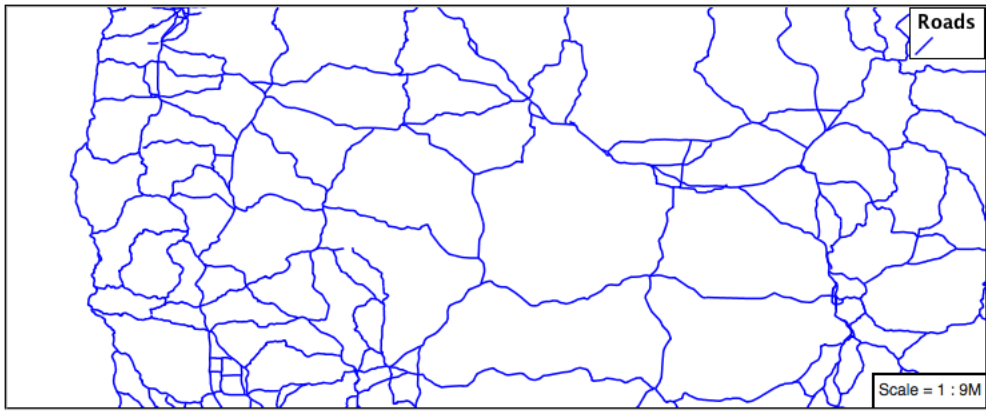
New style name:	`line_example`
Workspace for new layer:	Leave blank
Format:	`MBStyle`

3. Fill in the style editor

```
{
  "version": 8,
  "name": "line_example",
  "layers": [
      {
          "id": "line_example",
          "source-layer": "ne:roads",
          "type": "line",
      }
  ]
}
```

4. Click *Apply*
5. Click *Layer Preview* to see your new style applied to a layer.

   You can use this tab to follow along as the style is edited, it will refresh each time *Apply* is pressed.



6. You can see the equivalent SLD by requesting `http://localhost:8080/geoserver/rest/styles/line_example.sld?pretty=true` which will currently show the default line symbolizer we created.

```
<?xml version="1.0" encoding="UTF-8"?>
<sld:StyledLayerDescriptor xmlns="http://www.opengis.net/sld" xmlns:sld="http://www.opengis.net/sld" xmlns:gml="http://www.opengis.net/gml" xmlns:ogc="http://www.opengis.net/ogc" version="1.0.0">
  <sld:NamedLayer>
    <sld:Name>line_example</sld:Name>
    <sld:UserStyle>
      <sld:Name>line_example</sld:Name>
      <sld:FeatureTypeStyle>
        <sld:Name>name</sld:Name>
        <sld:Rule>
          <sld:LineSymbolizer/>
        </sld:Rule>
      </sld:FeatureTypeStyle>
    </sld:UserStyle>
  </sld:NamedLayer>
</sld:StyledLayerDescriptor>
```

We only specified the line layer, so all of the boilerplate arround was generated for us.

1. Additional properties cane be used fine-tune appearance. Use **line-color** to specify the colour and width of the line.

```
{
  "paint": {
    "line-color": "blue"
  }
}
```

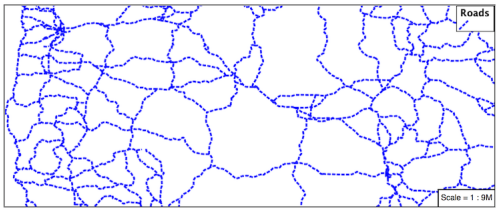
2. **line-width** lets us make the line wider

```
{
  "paint": {
    "line-color": "blue",
    "line-width": 2
  }
}
```

3. **line-dasharray** applies a dot dash pattern.

```
{
  "paint": {
    "line-color": "blue",
    "line-width": 2,
    "line-dasharray": [5, 2]
  }
}
```

4. Check the *Map* tab to preview the result.



Multiple Layers

Providing two strokes is often used to provide a contrasting edge (called casing) to thick lines. This can be created using two layers.

1. Start by filling in a bit of boilerplate that we'll be using

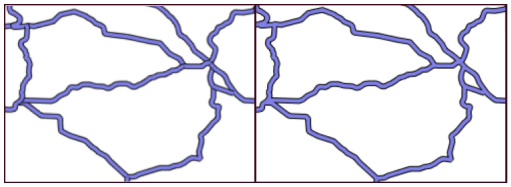
```
{
  "version": 8,
  "name": "line_example",
  "layers": [
    {
      "id": "line_example",
      "source-layer": "ne:roads",
      "type": "line",
      "paint": {
        "line-color": "#8080E6",
        "line-width": 3,
      }
    }
  ]
}
```

2. Add a second layer to the rule

```
{
  "version": 8,
  "name": "line_example",
  "layers": [
    {
      "id": "line_casing",
      "source-layer": "ne:roads",
      "type": "line",
      "paint": {
```

```
        "line-color": "black",
        "line-width": 5,
      }
    },
    {
      "id": "line_center",
      "source-layer": "ne:roads",
      "type": "line",
      "paint": {
        "line-color": "#8080E6",
        "line-width": 3,
      }
    }
  ]
}
```

The wider black line is first so it is drawn first, then the thinner blue line drawn second and so over top of the black line. This is called the painter's algorithm.



Label

Our next example is significant as it introduces how text labels are generated.

Fig. 6.313: Use of Label Property

This is also our first example making use of a dynamic style (where a value comes from an attribute from your data).

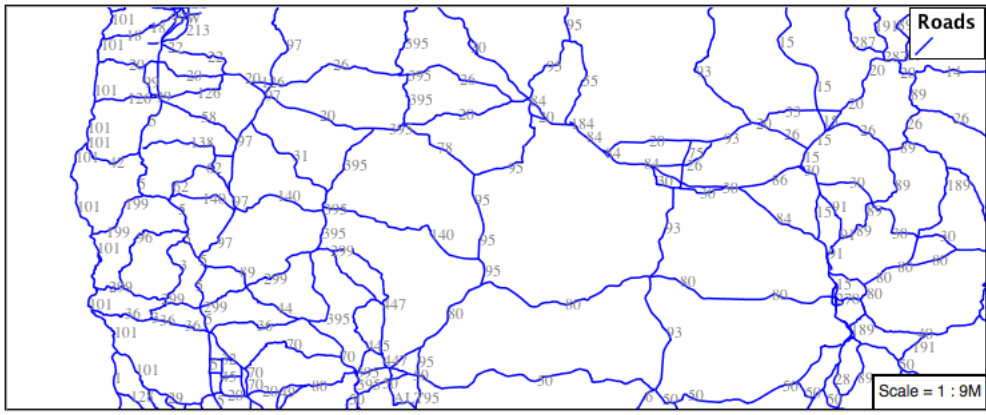
1. To enable LineString labeling we add a `symbol` layer with a `text-field`.

   Update `line_example` with the following:

```
{
  "version": 8,
  "name": "line_example",
  "layers": [
    {
      "id": "line",
      "source-layer": "ne:roads",
      "type": "line",
      "paint": {
        "line-color": "blue",
        "line-width": 1,
      }
    },
```

```
    {
      "id": "label",
      "source-layer": "ne:roads",
      "type": "symbol",
      "layout": {
        "text-field": "{name}"
      }
    }
  ]
}
```

2. The SLD standard documents the default label position for each kind of Geometry. For LineStrings the initial label is positioned on the midway point of the line.



3. We have used a feature property calculate a value for the label. The **label** is generated dynamically from the `name` attribute. Feature properties are supplied within curly braces, and must match the name of a property of the feature type.

```
{
 "version": 8,
 "name": "line_example",
 "layers": [
   {
     "id": "line",
     "source-layer": "ne:roads",
     "type": "line",
     "paint": {
       "line-color": "blue",
       "line-width": 1,
     }
   },
   {
     "id": "label",
     "source-layer": "ne:roads",
     "type": "symbol",
     "layout": {
       "text-field": "{name}"
     }
   }
 ]
}
```

4. Additional keys can be supplied to fine-tune label presentation:

```
{
  "version": 8,
  "name": "line_example",
  "layers": [
    {
      "id": "line",
      "source-layer": "ne:roads",
      "type": "line",
      "paint": {
        "line-color": "blue",
        "line-width": 1,
      }
    },
    {
      "id": "label",
      "source-layer": "ne:roads",
      "type": "symbol",
      "layout": {
        "text-field": "{name}",
        "symbol-placement": "line",
        "text-offset": [0, -8]
      }
      "paint": {
        "text-color": "black"
      }
    }
  ]
}
```

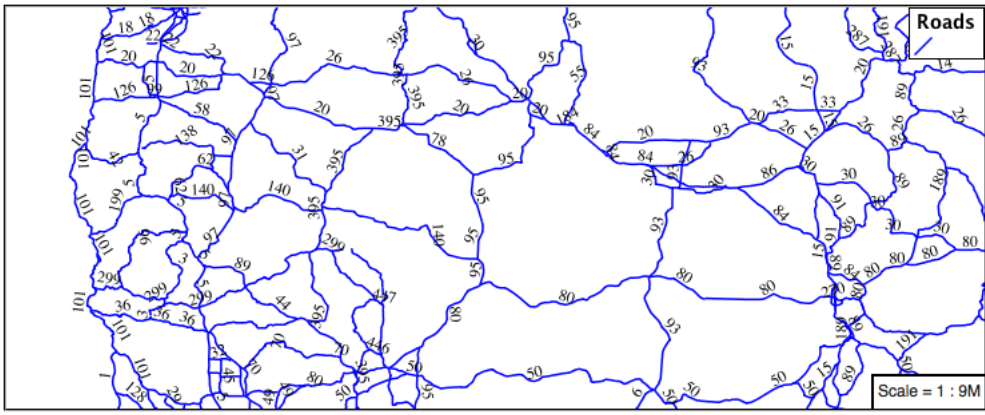
5. The **text-color** property is set to `black` to provide the colour of the text. Notice how this is a **paint** property, unlike all the others which are **layout** properties.

```
{
  "version": 8,
  "name": "line_example",
  "layers": [
    {
      "id": "line",
      "source-layer": "ne:roads",
      "type": "line",
      "paint": {
        "line-color": "blue",
        "line-width": 1,
      }
    },
    {
      "id": "label",
      "source-layer": "ne:roads",
      "type": "symbol",
      "layout": {
        "text-field": "{name}",
        "symbol-placement": "line",
        "text-offset": [0, -8]
      }
      "paint": {
```

```
        "text-color": "black"
      }
    }
  ]
}
```

6. The **symbol-placement** property is used to set how the label is placed with respect to the line. By default it is `point` which casues the label to be placed next to the midpoint as it would be for a point feature. When set to `line` it is placed along the line instead. **text-offset** specifies how far from the anchor the label should be placed, in both the x and y directions.

```
{
  "version": 8,
  "name": "line_example",
  "layers": [
    {
      "id": "line",
      "source-layer": "ne:roads",
      "type": "line",
      "paint": {
        "line-color": "blue",
        "line-width": 1,
      }
    },
    {
      "id": "label",
      "source-layer": "ne:roads",
      "type": "symbol",
      "layout": {
        "text-field": "{name}",
        "symbol-placement": "line",
        "text-offset": [0, -8]
      }
      "paint": {
        "text-color": "black"
      }
    }
  ]
}
```



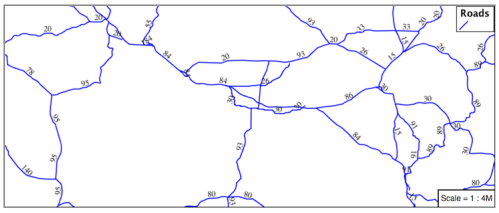
How Labeling Works

The rendering engine collects all the generated labels during the rendering of each layer. Then, during labeling, the engine sorts through the labels performing collision avoidance (to prevent labels overlapping). Finally the rendering engine draws the labels on top of the map. Even with collision avoidance you can spot areas where labels are so closely spaced that the result is hard to read.

1. The parameter **text-padding** provides additional space around our label for use in collision avoidance.

```
{
  "version": 8,
  "name": "line_example",
  "layers": [
    {
      "id": "line",
      "source-layer": "ne:roads",
      "type": "line",
      "paint": {
        "line-color": "blue",
        "line-width": 1,
      }
    },
    {
      "id": "label",
      "source-layer": "ne:roads",
      "type": "symbol",
      "layout": {
        "text-field": "{name}",
        "symbol-placement": "line",
        "text-offset": [0, -8],
        "text-padding": "10"
      }
      "paint": {
        "text-color": "black"
      }
    }
  ]
}
```

2. Each label is now separated from its neighbor, improving legibility.



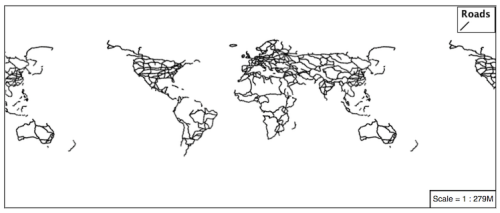
Zoom

This section explores the use of rules with filters and zoom restrictions.

1. Replace the *line_example* MBStyle definition with:

```
{
  "version": 8,
  "name": "line_example",
  "layers": [
    {
      "id": "line_example",
      "source-layer": "ne:roads",
      "type": "line",
      "filter": ["<", "scalerank", 4],
      "paint": {
        "line-color": "black",
        "line-width": 1
      }
    }
  ]
}
```

2. And use the *Map* tab to preview the result.



3. The **scalerank** attribute is provided by the Natural Earth dataset to allow control of the level of detail based on scale. Our filter short-listed all content with scalerank 4 or lower, providing a nice quick preview when we are zoomed out.

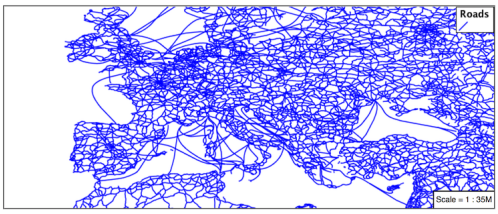
4. In addition to testing feature attributes, selectors can also be used to check the state of the rendering engine.

   Replace your MBStyle with the following:

```
{
  "version": 8,
  "name": "line_example",
  "layers": [
    {
      "id": "line_black",
      "source-layer": "ne:roads",
      "type": "line",
      "maxzoom": 3,
      "paint": {
        "line-color": "black",
        "line-width": 1
      }
    },
    {
      "id": "line_blue",
      "source-layer": "ne:roads",
      "type": "line",
      "minzoom": 3,
      "paint": {
```

```
        "line-color": "blue",
        "line-width": 1
      }
    }
  ]
}
```

5. As you adjust the scale in the *Map* preview (using the mouse scroll wheel) the color will change between black and blue. You can read the current scale in the bottom right corner, and the legend will change to reflect the current style.



6. Putting these two ideas together allows control of level detail based on scale:

```
{
  "version": 8,
  "name": "line_example",
  "layers": [
    {
      "id": "line_else",
      "source-layer": "ne:roads",
      "type": "line",
      "filter": [">", "scalerank", 7],
      "minzoom": 7,
      "paint": {
        "line-color": "#888888",
        "line-width": 1
      }
    },
    {
      "id": "line_7",
      "source-layer": "ne:roads",
      "type": "line",
      "filter": ["==", "scalerank", 7],
      "minzoom": 6,
      "paint": {
        "line-color": "#777777",
        "line-width": 1
      }
    },
    {
      "id": "line_6",
      "source-layer": "ne:roads",
      "type": "line",
      "filter": ["==", "scalerank", 6],
      "minzoom": 5,
      "paint": {
        "line-color": "#444444",
        "line-width": 1
```

```
      }
    },
    {
      "id": "line_5_1",
      "source-layer": "ne:roads",
      "type": "line",
      "filter": ["==", "scalerank", 5],
      "minzoom": 4,
      "maxzoom": 7
      "paint": {
        "line-color": "#000055",
        "line-width": 1
      }
    },
    {
      "id": "line_5_2",
      "source-layer": "ne:roads",
      "type": "line",
      "filter": ["==", "scalerank", 5],
      "minzoom": 7,
      "paint": {
        "line-color": "#000055",
        "line-width": 2
      }
    },
    {
      "id": "line_5_1",
      "source-layer": "ne:roads",
      "type": "line",
      "filter": ["<=", "scalerank", 4],
      "maxzoom": 5,
      "paint": {
        "line-color": "black",
        "line-width": 1
      }
    },
    {
      "id": "line_5_2",
      "source-layer": "ne:roads",
      "type": "line",
      "filter": ["<=", "scalerank", 4],
      "minzoom": 5,
      "maxzoom": 7
      "paint": {
        "line-color": "black",
        "line-width": 2
      }
    },
    {
      "id": "line_5_4",
      "source-layer": "ne:roads",
      "type": "line",
      "filter": ["<=", "scalerank", 4],
      "minzoom": 7,
      "paint": {
        "line-color": "black",
        "line-width": 4
      }
```

```
        }
      ]
    }
```

7. When a rule has both a filter and a scale, it will trigger when both are true.

Bonus

Finished early? Here are some opportunities to explore what we have learned, and extra challenges requiring creativity and research.

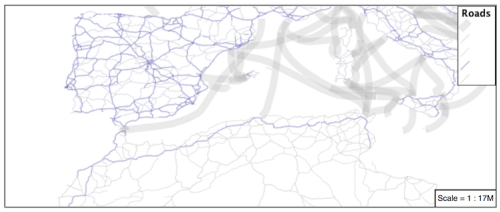
In a classroom setting please divide the challenges between teams (this allows us to work through all the material in the time available).

Challenge Classification

1. The roads **type** attribute provides classification information.

   You can **Layer Preview** to inspect features to determine available values for type.

2. **Challenge:** Create a new style adjust road appearance based on **type**.



Note: The available values are 'Major Highway','Secondary Highway','Road' and 'Unknown'.

Note: Answer *provided* at the end of the workbook.

Challenge One Rule Classification

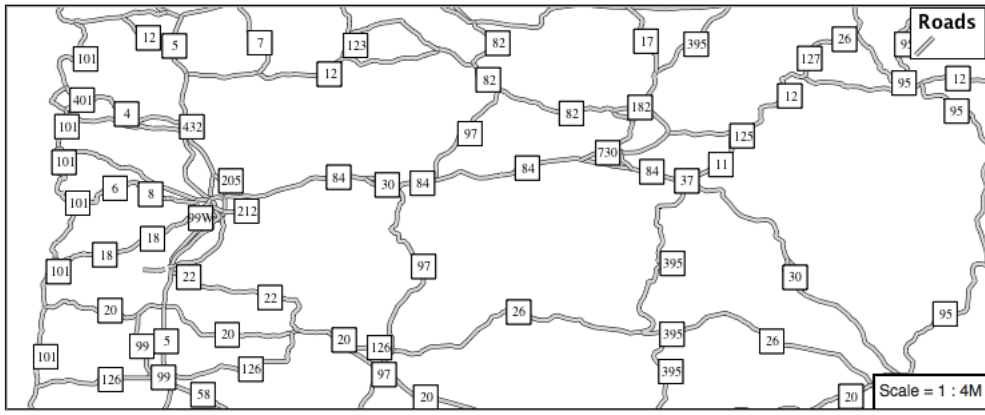
1. You can save a lot of typing by doing your classification in an expression using arithmetic or the `Recode` function

2. **Challenge:** Create a new style and classify the roads based on their scale rank using expressions in a single rule instead of multiple rules with filters.

Note: Answer *provided* at the end of the workbook.

Challenge Label Shields

1. The traditional presentation of roads in the US is the use of a shield symbol, with the road number marked on top.
2. *Challenge:* Have a look at the documentation for putting a graphic on a text symbolizer in SLD and reproduce this technique in MBStyle.



Note: Answer *provided* at the end of the workbook.

Polygons

Next we look at how MBStyle styling can be used to represent polygons.

Fig. 6.314: Polygon Geometry

Review of polygon symbology:

- Polygons offer a direct representation of physical extent or the output of analysis.
- The visual appearance of polygons reflects the current scale.
- Polygons are recorded as a LinearRing describing the polygon boundary. Further LinearRings can be used to describe any holes in the polygon if present.

  The Simple Feature for SQL Geometry model (used by GeoJSON) represents these areas as Polygons, the ISO 19107 geometry model (used by GML3) represents these areas as Surfaces.

- SLD uses a **PolygonSymbolizer** to describe how the shape of a polygon is drawn. The primary characteristic documented is the **Fill** used to shade the polygon interior. The use of a **Stroke** to describe the polygon boundary is optional.
- Labeling of a polygon is anchored to the centroid of the polygon. GeoServer provides a vendor-option to allow labels to line wrap to remain within the polygon boundaries.

  For our Polygon exercises we will try and limit our MBStyle documents to a single rule, in order to showcase the properties used for rendering.

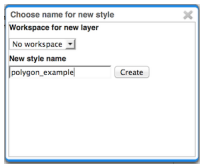
  Reference:

- *MBStyle Reference*
- MapBox Style Spec Fill Layer
- *Polygons* (User Manual | SLD Reference)

This exercise makes use of the `ne:states_provinces_shp` layer.

1. Navigate to *Styles*.
2. Create a new style `polygon_example`.

Name:	`polygon_example`
Workspace:	`No workspace`
Format:	`MBStyle`

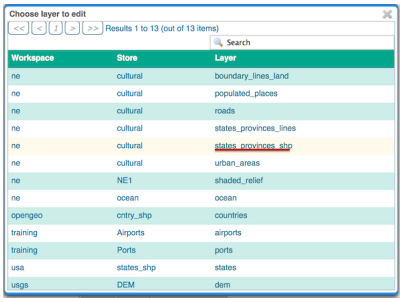


3. Enter the following style and click :menuselection:`Apply` to save:

```
{
  "version": 8,
  "name": "polygon_example",
  "layers": [
    {
      "id": "polygon_example",
      "source-layer": "ne:states_provinces_shp",
      "type": "fill",
      "paint": {
        "fill-color": "lightgrey"
      }
    }
  ]
}
```

4. Click on the tab *Layer Preview* to preview.

5. Set `ne:states_provinces_shp` as the preview layer.



Fill and Outline

The **fill** layer controls the display of polygon data.

The **fill-color** property is used to provide the color used to draw the interior of a polygon.

1. Replace the contents of `polygon_example` with the following **fill** example:

```
{
  "version": 8,
  "name": "polygon_example",
  "layers": [
    {
      "id": "polygon_example",
      "source-layer": "ne:states_provinces_shp",
      "type": "fill",
      "paint": {
        "fill-color": "gray"
      }
    }
  ]
}
```

2. The *Map* tab can be used preview the change:

3. To draw the boundary of the polygon the **fill-outline** property is used:

   The **fill-outline** property is used to provide the color of the polygon boundary. For more advanced boundary styling, use a seperate line layer.

```
{
  "version": 8,
  "name": "polygon_example",
  "layers": [
    {
      "id": "polygon_example",
      "source-layer": "ne:states_provinces_shp",
      "type": "fill",
      "paint": {
        "fill-color": "gray",
        "fill-outline-color": "black"
      }
    }
  ]
}
```

   Note: Technically the boundary of a polygon is a specific case of a LineString where the first and last vertex are the same, forming a closed LinearRing.

4. The effect of adding **fill-outline** is shown in the map preview:

5. An interesting technique when styling polygons in conjunction with background information is to control the fill opacity.

   The **fill-opacity** property is used to adjust transparency (provided as range from 0.0 to 1.0). Use of **fill-opacity** to render polygons works well in conjunction with a raster base map. This approach allows details of the base map to shown through. **fill-opacity** affects both the fill and the fill outline.

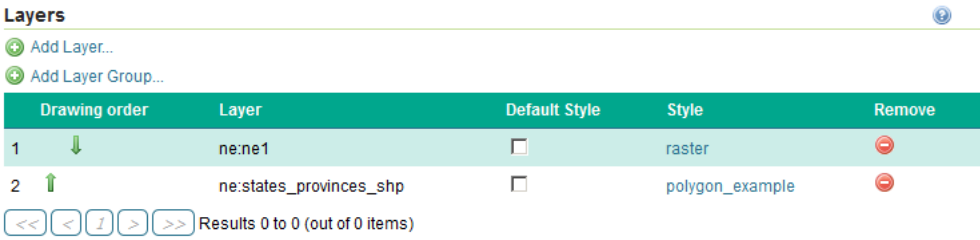
   The **stroke-opacity** property is used in a similar fashion, as a range from 0.0 to 1.0.

```
{
  "version": 8,
  "name": "polygon_example",
  "layers": [
    {
      "id": "polygon_example",
      "source-layer": "ne:states_provinces_shp",
      "type": "fill",
```

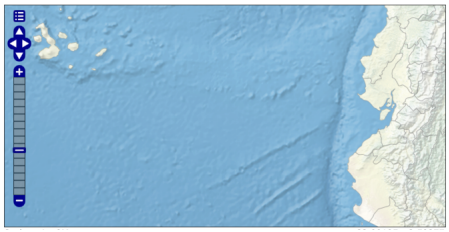
```
                "paint": {
                  "fill-opacity": 0.5,
                  "fill-color": "white",
                  "fill-outline-color": "lightgrey"
                }
              }
            ]
          }
```

6. As shown in the map preview:

7. This effect can be better appreciated using a layer group.



Where the transparent polygons is used lighten the landscape provided by the base map.



Pattern

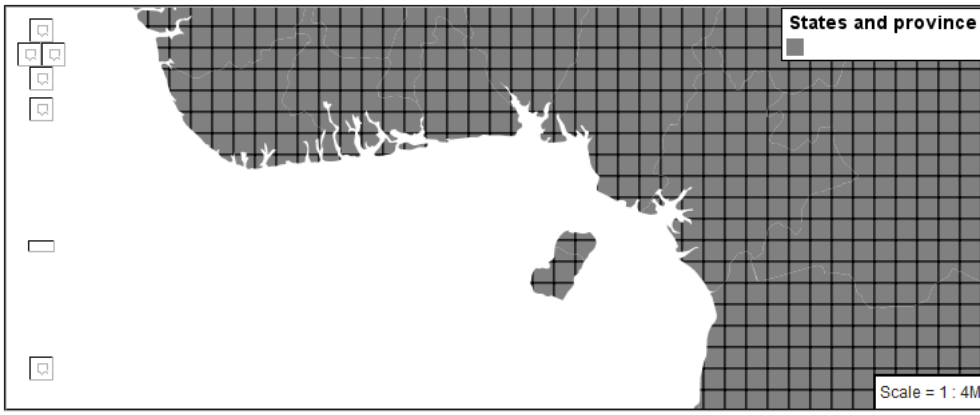
The **fill-pattern** property can be used to provide a pattern.

The fill pattern is defined by repeating an image defined in a spritesheet.

1. Update *polygon_example* with the following sprite as a repeating fill pattern:

```
{
  "version": 8,
  "name": "polygon_example",
  "sprite
↪": "http://localhost:8080/geoserver/styles/sprites"
  "layers": [
    {
      "id": "polygon_example",
      "source-layer": "ne:states_provinces_shp",
      "type": "fill",
      "paint": {
        "fill-pattern": "grey_square16"
      }
    }
  ]
}
```

2. The map preview (and legend) will show the result:



3. You can view the names of all the icons in the spritesheet by looking at its json definition, at http://localhost:8080/geoserver/styles/sprites.json.

```
{
    "white_square16": {
        "height": 16,
        "pixelRatio": 1,
        "width": 16,
        "x": 1,
        "y": 1
    },
    "grey_square8": {
        "height": 8,
        "pixelRatio": 1,
        "width": 8,
        "x": 24,
        "y": 18
    },
    "grey_square16": {
        "height": 16,
```

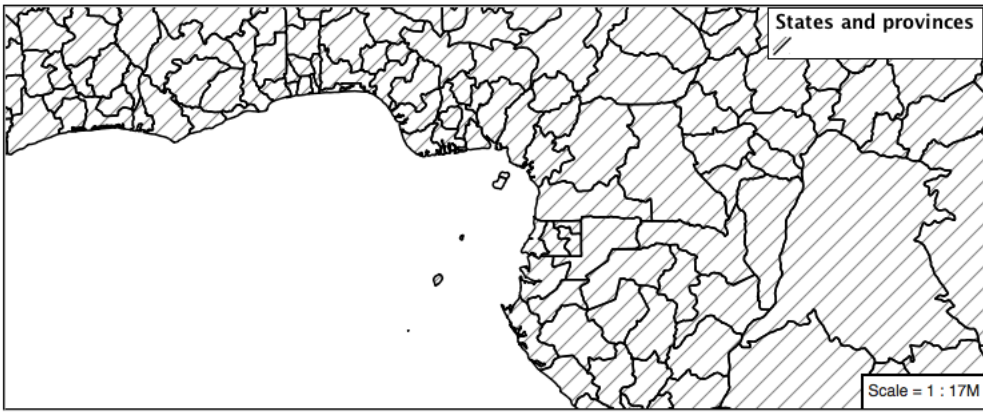
```
            "pixelRatio": 1,
            "width": 16,
            "x": 18,
            "y": 1
        },
        "grey_square22": {
            "height": 22,
            "pixelRatio": 1,
            "width": 22,
            "x": 1,
            "y": 18
        },
        "green_square16": {
            "height": 16,
            "pixelRatio": 1,
            "width": 16,
            "x": 35,
            "y": 1
        },
        "grey_x": {
            "height": 30,
            "pixelRatio": 1,
            "width": 30,
            "x": 1,
            "y": 41
        },
        "grey_diag8": {
            "height": 8,
            "pixelRatio": 1,
            "width": 8,
            "x": 24,
            "y": 27
        },
        "grey_diag16": {
            "height": 16,
            "pixelRatio": 1,
            "width": 16,
            "x": 35,
            "y": 18
        },
        "grey_circle": {
            "height": 17,
            "pixelRatio": 1,
            "width": 17,
            "x": 36,
            "y": 36
        },
        "airport": {
            "height": 16,
            "pixelRatio": 1,
            "width": 16,
            "x": 52,
            "y": 18
        },
        "port": {
            "height": 16,
            "pixelRatio": 1,
            "width": 16,
```

```
            "x": 52,
            "y": 1
        },
        "star": {
            "height": 16,
            "pixelRatio": 1,
            "width": 16,
            "x": 69,
            "y": 1
        }
    }
```

Update the example to use **grey_diag16** for a pattern of left hatching.

```
{
  "version": 8,
  "name": "polygon_example",
  "sprite
↪": "http://localhost:8080/geoserver/styles/sprites"
  "layers": [
    {
      "id": "polygon_example",
      "source-layer": "ne:states_provinces_shp",
      "type": "fill",
      "paint": {
        "fill-pattern": "grey_diag16"
      }
    }
  ]
}
```

4. This approach is well suited to printed output or low color devices.

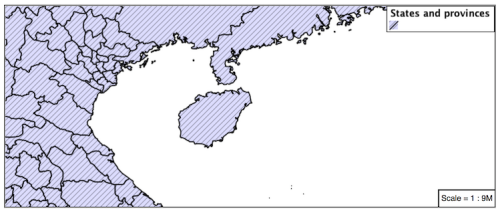


5. Multiple fills can be applied by using a seperate layer for each fill.

```
{
  "version": 8,
  "name": "polygon_example",
  "sprite
↪": "http://localhost:8080/geoserver/styles/sprites"
  "layers": [
    {
```

```
      "id": "polygon_background",
      "source-layer": "ne:states_provinces_shp",
      "type": "fill",
      "paint": {
        "fill-color": "#DDDDFF",
        "fill-outline-color": "black"
      }
    },
    {
      "id": "polygon_pattern",
      "source-layer": "ne:states_provinces_shp",
      "type": "fill",
      "paint": {
        "fill-pattern": "grey_diag8"
      }
    }
  ]
}
```

6. The resulting image has a solid fill, with a pattern drawn overtop.



Label

Labeling polygons follows the same approach used for LineStrings.

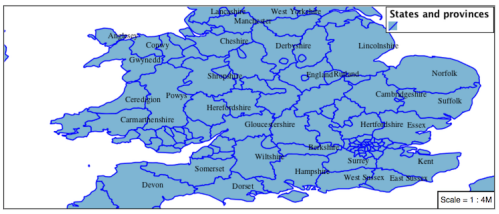
1. By default labels are drawn starting at the centroid of each polygon.

2. Try out **text-field** and **text-color** by replacing our `polygon_example` with the following:

```
{
  "version": 8,
  "name": "polygon_example",
  "layers": [
    {
      "id": "polygon_fill",
      "source-layer": "ne:states_provinces_shp",
      "type": "fill",
      "paint": {
        "fill-color": "#7EB5D3",
        "fill-outline-color": "blue"
      }
    },
    {
      "id": "polygon_label",
      "source-layer": "ne:states_provinces_shp",
```

```
      "type": "symbol",
      "layout": {
        "text-field": "{name}"
      },
      "paint": {
        "text-color": "black"
      }
    }
  ]
}
```

3. Each label is drawn from the lower-left corner as shown in the `Map` preview.



4. We can adjust how the label is drawn at the polygon centroid.

   The property **text-anchor** provides two numbers expressing how a label is aligned with respect to the centroid. Adjusting the **text-anchor** is the recommended approach to positioning your labels.

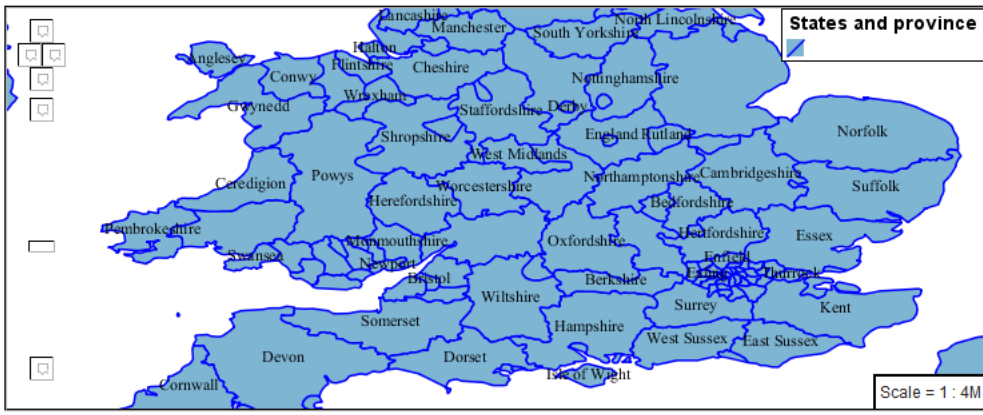
5. Using the **text-anchor** property we can center our labels with respect to geometry centroid.

   To align the center of our label we select "center" below:

```
{
  "version": 8,
  "name": "polygon_example",
  "layers": [
    {
      "id": "polygon_fill",
      "source-layer": "ne:states_provinces_shp",
      "type": "fill",
      "paint": {
        "fill-color": "#7EB5D3",
        "fill-outline-color": "blue"
      }
    },
    {
      "id": "polygon_label",
      "source-layer": "ne:states_provinces_shp",
      "type": "symbol",
      "layout": {
        "text-field": "{name}",
        "text-anchor": "center"
      },
      "paint": {
        "text-color": "black"
      }
```

```
        }
      ]
    }
```

6. The labeling position remains at the polygon centroid. We adjust alignment by controlling which part of the label we are "snapping" into position.

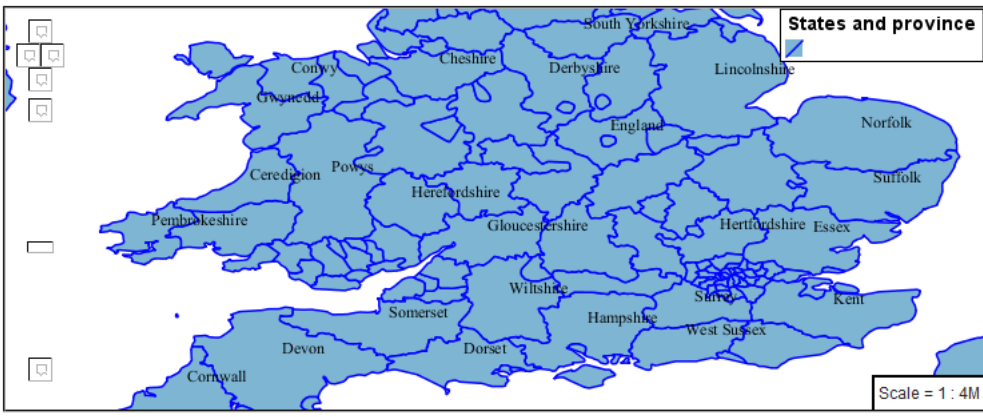


7. The property **text-translate** can be used to provide an initial displacement using and x and y offset.

8. This offset is used to adjust the label position relative to the geometry centroid resulting in the starting label position.

```
{
  "version": 8,
  "name": "polygon_example",
  "layers": [
    {
      "id": "polygon_fill",
      "source-layer": "ne:states_provinces_shp",
      "type": "fill",
      "paint": {
        "fill-color": "#7EB5D3",
        "fill-outline-color": "blue"
      }
    },
    {
      "id": "polygon_label",
      "source-layer": "ne:states_provinces_shp",
      "type": "symbol",
      "layout": {
        "text-field": "{name}",
      },
      "paint": {
        "text-color": "black",
        "text-translate": [0, -7]
      }
    }
  ]
}
```

9. Confirm this result in the map preview.



10. These two settings can be used together.

    The rendering engine starts by determining the label position generated from the geometry centroid and the **text-translate** displacement. The bounding box of the label is used with the **text-anchor** setting align the label to this location.

    Step 1: starting label position = centroid + displacement

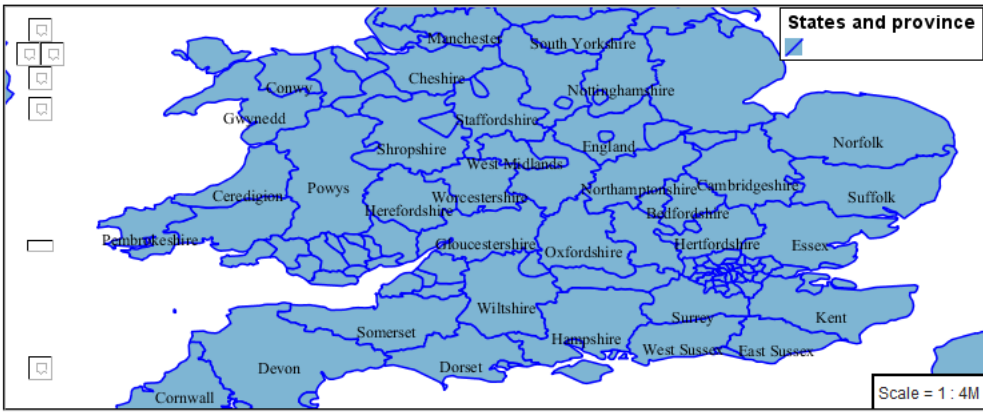
    Step 2: snap the label anchor to the starting label position

11. To move our labels down (allowing readers to focus on each shape) we can use displacement combined with followed by horizontal alignment.

```
{
  "version": 8,
  "name": "polygon_example",
  "layers": [
    {
      "id": "polygon_fill",
      "source-layer": "ne:states_provinces_shp",
      "type": "fill",
      "paint": {
        "fill-color": "#7EB5D3",
        "fill-outline-color": "blue"
      }
    },
    {
      "id": "polygon_label",
      "source-layer": "ne:states_provinces_shp",
      "type": "symbol",
      "layout": {
        "text-field": "{name}",
        "text-anchor": "left"
      },
      "paint": {
        "text-color": "black",
        "text-translate": [0, -7]
      }
    }
```

```
    ]
  }
```

12. As shown in the map preview.



Legibility

When working with labels a map can become busy very quickly, and difficult to read.

1. MBStyle extensive proterties for controlling the labelling process.

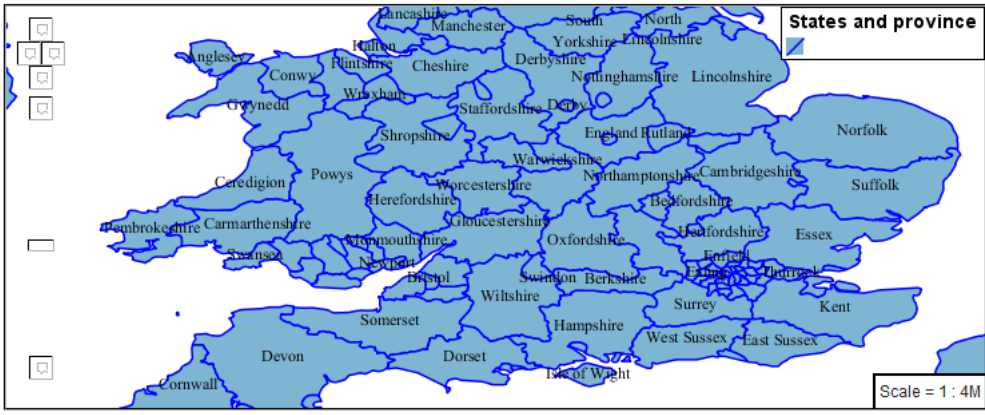
   One common property for controlling labeling is **text-max-width**, which allows any labels extending past the provided width will be wrapped into multiple lines.

2. Using this we can make a small improvement in our example:

```
{
  "version": 8,
  "name": "polygon_example",
  "layers": [
    {
      "id": "polygon_fill",
      "source-layer": "ne:states_provinces_shp",
      "type": "fill",
      "paint": {
        "fill-color": "#7EB5D3",
        "fill-outline-color": "blue"
      }
    },
    {
      "id": "polygon_label",
      "source-layer": "ne:states_provinces_shp",
      "type": "symbol",
      "layout": {
        "text-field": "{name}",
        "text-anchor": "center"
        "text-max-width": 14
      },
      "paint": {
        "text-color": "black",
```

```
        }
      }
    ]
  }
```

3. As shown in the following preview.

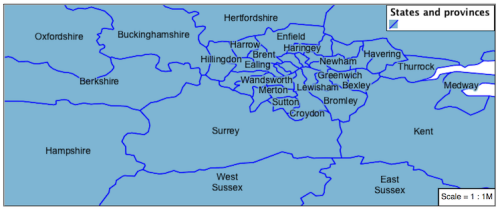


4. Even with this improved spacing between labels, it is difficult to read the result against the complicated line work.

   Use of a halo to outline labels allows the text to stand out from an otherwise busy background. In this case we will make use of the fill color, to provide some space around our labels. We will also change the font to Arial.

```
{
  "version": 8,
  "name": "polygon_example",
  "layers": [
    {
      "id": "polygon_fill",
      "source-layer": "ne:states_provinces_shp",
      "type": "fill",
      "paint": {
        "fill-color": "#7EB5D3",
        "fill-outline-color": "blue"
      }
    },
    {
      "id": "polygon_label",
      "source-layer": "ne:states_provinces_shp",
      "type": "symbol",
      "layout": {
        "text-field": "{name}",
        "text-anchor": "center"
        "text-max-width": 14,
        "text-font": ["Arial"]
      },
      "paint": {
        "text-color": "black",
        "text-halo-color": "#7EB5D3",
        "text-halo-width": 2
```

```
        }
      }
    ]
  }
```



Theme

A thematic map (rather than focusing on representing the shape of the world) uses elements of style to illustrate differences in the data under study. This section is a little more advanced and we will take the time to look at the generated SLD file.

1. We can use a site like ColorBrewer to explore the use of color theming for polygon symbology. In this approach the the fill color of the polygon is determined by the value of the attribute under study.

   This presentation of a dataset is known as "theming" by an attribute.

2. For our `ne:states_provinces_shp` dataset, a **mapcolor9** attribute has been provided for this purpose. Theming by **mapcolor9** results in a map where neighbouring countries are visually distinct.

Qualitative 9-class Set3		
#8dd3c7	#fb8072	#b3de69
#ffffb3	#80b1d3	#fccde5
#bebada	#fdb462	#d9d9d9

   If you are unfamiliar with theming you may wish to visit http://colorbrewer2.org to learn more. The **i** icons provide an adequate background on theming approaches for qualitative, sequential and diverging datasets.

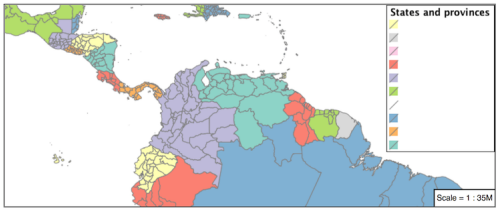
3. The first approach we will take is to directly select content based on **colormap**, providing a color based on the **9-class Set3** palette above:

```
{
  "version": 8,
  "name": "polygon_example",
  "layers": [
    {
      "id": "polygon_1",
      "filter": ["==", "mapcolor9", 1],
      "source-layer": "ne:states_provinces_shp",
      "type": "fill",
      "paint": {
        "fill-color": "#8DD3C7",
```

```
      "fill-outline-color": "gray"
    }
  },
  {
    "id": "polygon_2",
    "filter": ["==", "mapcolor9", 2],
    "source-layer": "ne:states_provinces_shp",
    "type": "fill",
    "paint": {
      "fill-color": "#FFFFB3",
      "fill-outline-color": "gray"
    }
  },
  {
    "id": "polygon_3",
    "filter": ["==", "mapcolor9", 3],
    "source-layer": "ne:states_provinces_shp",
    "type": "fill",
    "paint": {
      "fill-color": "#BEBADA",
      "fill-outline-color": "gray"
    }
  },
  {
    "id": "polygon_4",
    "filter": ["==", "mapcolor9", 4],
    "source-layer": "ne:states_provinces_shp",
    "type": "fill",
    "paint": {
      "fill-color": "#FB8072",
      "fill-outline-color": "gray"
    }
  },
  {
    "id": "polygon_5",
    "filter": ["==", "mapcolor9", 5],
    "source-layer": "ne:states_provinces_shp",
    "type": "fill",
    "paint": {
      "fill-color": "#80B1D3",
      "fill-outline-color": "gray"
    }
  },
  {
    "id": "polygon_6",
    "filter": ["==", "mapcolor9", 6],
    "source-layer": "ne:states_provinces_shp",
    "type": "fill",
    "paint": {
      "fill-color": "#FDB462",
      "fill-outline-color": "gray"
    }
  },
  {
    "id": "polygon_7",
    "filter": ["==", "mapcolor9", 7],
    "source-layer": "ne:states_provinces_shp",
    "type": "fill",
```

```
      "paint": {
        "fill-color": "#B3DE69",
        "fill-outline-color": "gray"
      }
    },
    {
      "id": "polygon_8",
      "filter": ["==", "mapcolor9", 8],
      "source-layer": "ne:states_provinces_shp",
      "type": "fill",
      "paint": {
        "fill-color": "#FCCDE5",
        "fill-outline-color": "gray"
      }
    },
    {
      "id": "polygon_9",
      "filter": ["==", "mapcolor9", 9],
      "source-layer": "ne:states_provinces_shp",
      "type": "fill",
      "paint": {
        "fill-color": "#D9D9D9",
        "fill-outline-color": "gray"
      }
    }
  ]
}
```

4. The *Map* tab can be used to preview this result.



5. Property functions can be used to make theming substantially easier, by directly mapping property values to style values using an array of stops. MBStyle supports three types of function interpolation, which is used to define the behavior between these stops:

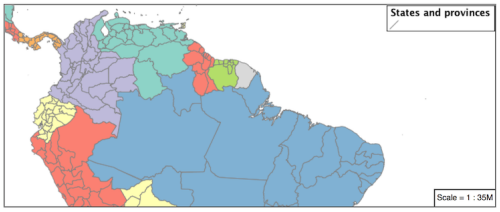
- **categorical**: Used the theme qualitative data. Attribute values are directly mapped to styling property such as **fill** or **stroke-width**. Equvalent to the SLD **Recode** function.
- **interval**: Used the theme quantitative data. Categories are defined using min and max ranges, and values are sorted into the appropriate category. Equvalent to the SLD **Categorize** function.
- **exponential**: Used to smoothly theme quantitative data by calculating a styling property based on an attribute value. Supports a **base** attribute for controlling the steepness of interpolation. When **base** is 1, this is equivalent to the SLD **Interpolate** function.

  Theming is an activity, producing a visual result allow map readers to learn more about how an attribute is distributed spatially. We are free to produce this visual in the most efficient way possible.

6. Swap out **mapcolor9** theme to use the **categorical** function:

```
{
  "version": 8,
  "name": "polygon_example",
  "layers": [
    {
      "id": "polygon",
      "source-layer": "ne:states_provinces_shp",
      "type": "fill",
      "paint": {
        "fill-color": {
          "property": "mapcolor9",
          "type": "categorical",
          "stops": [
            [1, "#8dd3c7"],
            [2, "#ffffb3"],
            [3, "#bebada"],
            [4, "#fb8072"],
            [5, "#80b1d3"],
            [6, "#fdb462"],
            [7, "#b3de69"],
            [8, "#fccde5"],
            [9, "#d9d9d9"]
          ]
        },
        "fill-outline-color": "gray"
      }
    }
  ]
}
```

7. The *Map* tab provides the same preview.



8. The *Generated SLD* tab shows where things get interesting. Our generated style now consists of a single **Rule**:

```
<sld:Rule>
   <sld:PolygonSymbolizer>
      <sld:Fill>
         <sld:CssParameter name="fill">
            <ogc:Function name="Recode">
               <ogc:PropertyName>mapcolor9</ogc:PropertyName>
               <ogc:Literal>1</ogc:Literal>
               <ogc:Literal>#8dd3c7</ogc:Literal>
               <ogc:Literal>2</ogc:Literal>
               <ogc:Literal>#ffffb3</ogc:Literal>
               <ogc:Literal>3</ogc:Literal>
               <ogc:Literal>#bebada</ogc:Literal>
               <ogc:Literal>4</ogc:Literal>
```

```
                                        <ogc:Literal>#fb8072</ogc:Literal>
                                    <ogc:Literal>5</ogc:Literal>
                                        <ogc:Literal>#80b1d3</ogc:Literal>
                                    <ogc:Literal>6</ogc:Literal>
                                        <ogc:Literal>#fdb462</ogc:Literal>
                                    <ogc:Literal>7</ogc:Literal>
                                        <ogc:Literal>#b3de69</ogc:Literal>
                                    <ogc:Literal>8</ogc:Literal>
                                        <ogc:Literal>#fccde5</ogc:Literal>
                                    <ogc:Literal>9</ogc:Literal>
                                        <ogc:Literal>#d9d9d9</ogc:Literal>
                                </ogc:Function>
                                </sld:CssParameter>
                            </sld:Fill>
                        </sld:PolygonSymbolizer>
                        <sld:LineSymbolizer>
                            <sld:Stroke>
                                <sld:CssParameter␣
                    ↪name="stroke">#808080</sld:CssParameter>
                                <sld:CssParameter␣
                    ↪name="stroke-width">0.5</sld:CssParameter>
                            </sld:Stroke>
                        </sld:LineSymbolizer>
                    </sld:Rule>
```

Bonus

The following optional explore and challenge activities offer a chance to review and apply the ideas introduced here. The challenge activities equire a bit of creativity and research to complete.

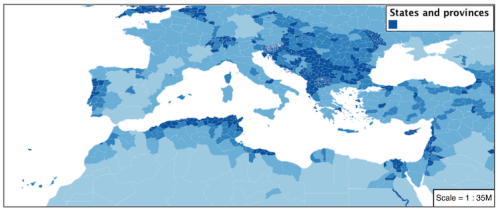
In a classroom setting you are encouraged to team up into groups, with each group taking on a different challenge.

Explore Interval

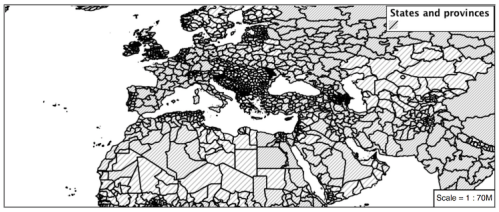
1. The **interval** function can be used to generate property values based on quantitative information. Here is an example using interval to color states according to size.

```
                    {
                     "version": 8,
                     "name": "polygon_example",
                     "layers": [
                       {
                         "id": "polygon",
                         "source-layer": "ne:states_provinces_shp",
                         "type": "fill",
                         "paint": {
                           "fill-color": {
                             "property": "Shape_Area",
                             "type": "interval",
                             "stops": [
                               [0, "#08519c"],
                               [0.5, "#3182bd"],
```

```
                                        [1, "#6baed6"],
                                        [5, "#9ecae1"],
                                        [60, "#c6dbef"],
                                        [80, "#eff3ff"]
                                    ]
                                }
                            }
                        }
                    ]
                }
```



2. An exciting use of the GeoServer **shape** symbols is the theming by changing the **size** used for pattern density.
3. **Explore:** Use the **interval** function to theme by **datarank**.



Note: Answer *provided* at the end of the workbook.

Challenge Halo

1. The halo example used the fill color and opacity for a muted halo, while this improved readability it did not bring attention to our labels.

   A common design choice for emphasis is to outline the text in a contrasting color.
2. **Challenge:** Produce a map that uses a white halo around black text.

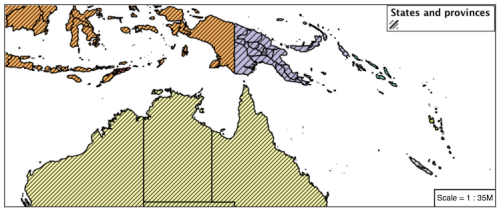
Note: Answer *provided* at the end of the workbook.

Challenge Theming using Multiple Attributes

1. A powerful tool is theming using multiple attributes. This is an important concept allowing map readers to perform "integration by

eyeball" (detecting correlations between attribute values information).

2. **Challenge:** Combine the **mapcolor9** and **datarank** examples to reproduce the following map.

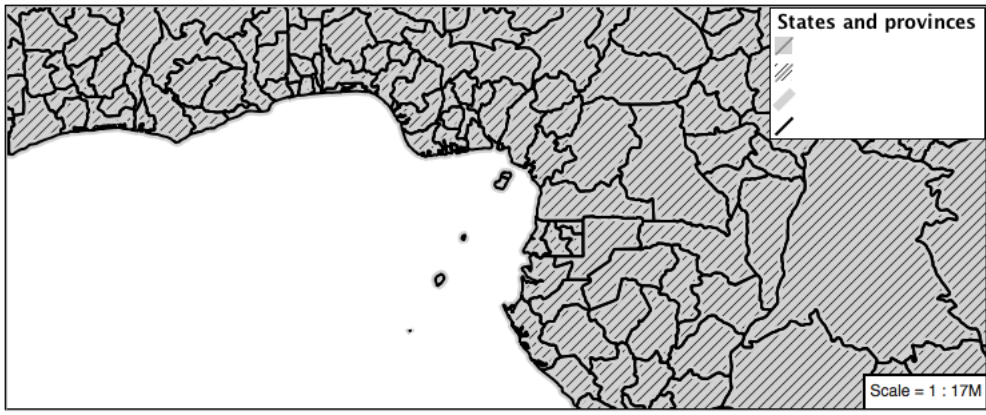


Note: Answer *provided* at the end of the workbook.

Challenge Use of Z-Index

1. Earlier we looked at using multiple **layers** to simulate line string casing. The line work was drawn twice, once with thick line, and then a second time with a thinner line. The resulting effect is similar to text halos - providing breathing space around complex line work allowing it to stand out.

2. **Challenge:** Use what you know of rendering order to reproduce the following map:



Note: Answer *provided* at the end of the workbook.

Points

The next stop of the mbstyle styling tour is the representation of points.

Review of point symbology:

- Points are used to represent a location only, and do not form a shape. The visual width of lines do not change depending on scale.
- SLD uses a **PointSymbolizer** record how the shape of a line is drawn.
- Labeling of points is anchored to the point location.

  As points have no inherent shape of of their own, emphasis is placed on marking locations with an appropriate symbol.

  Reference:

- *MBStyle Reference*
- MapBox Style Spec Symbol Layer
- MapBox Style Spec Circle Layer
- *Point* (User Manual | SLD Reference)

This exercise makes use of the `ne:populated_places` layer.

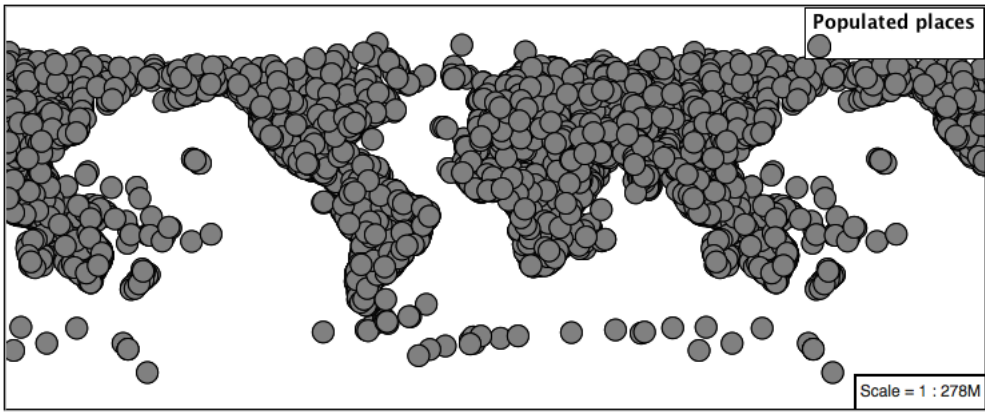
1. Navigate to the **Styles** page.
2. Click *Add a new style* and choose the following:

Name:	`point_example`
Workspace:	`No workspace`
Format:	`MBStyle`

3. Replace the initial MBStyle definition with the following and click *apply*:

```
{
  "version": 8,
  "name": "point_example",
  "sprite
↪": "http://localhost:8080/geoserver/styles/sprites"
  "layers": [
    {
      "id": "point_example",
      "type": "symbol",
      "source-layer": "ne:populated_places",
      "layout": {
        "icon-image": "grey_circle",
      }
    }
  ]
}
```

4. And use the *Layer Preview* tab to preview the result.



Sprite

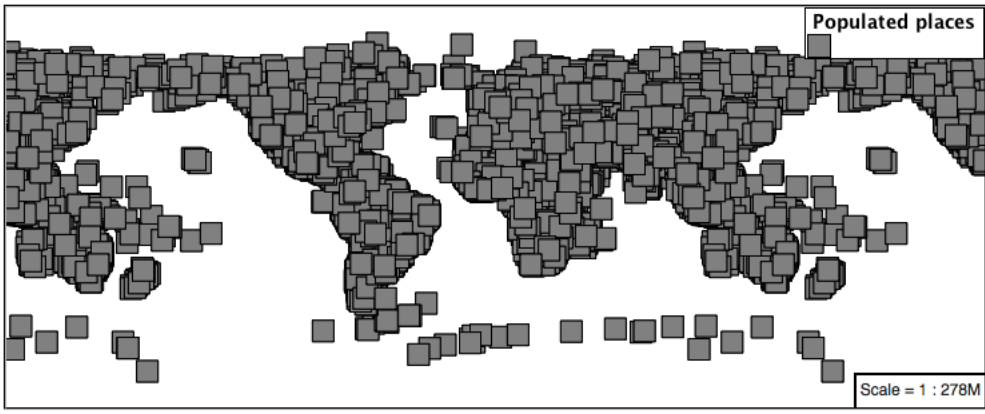
The **symbol** layer controls the display of point data. Points are typically represented with an **icon-image**.

MBStyle uses a spritesheet defined at the top-level of the style to define a set of icons. You can view the names of all the icons in the spritesheet by looking at its json definition, at http://localhost:8080/geoserver/styles/sprites.json.

1. Change the symbol used by the style to a square:

```
{
  "version": 8,
  "name": "point_example",
  "sprite
↪": "http://localhost:8080/geoserver/styles/sprites"
  "layers": [
    {
      "id": "point_example",
      "type": "symbol",
      "source-layer": "ne:populated_places",
      "layout": {
        "icon-image": "grey_square16",
      }
    }
  ]
}
```

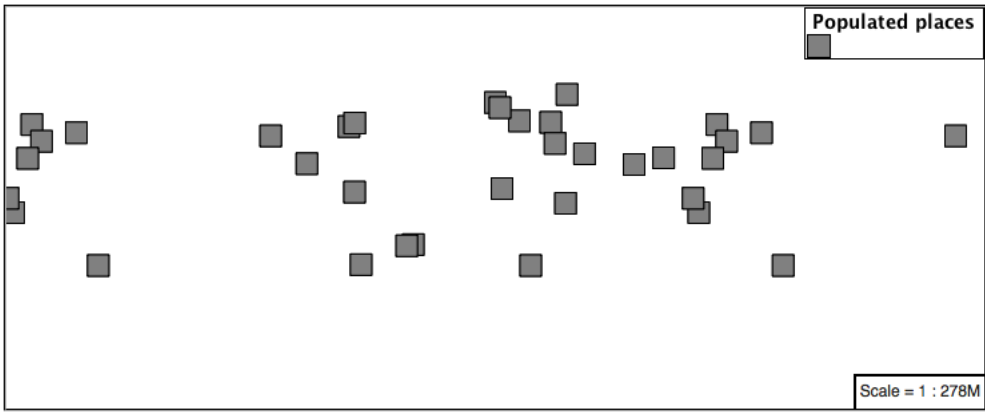
2. Map Preview:



3. Before we continue we will use a selector to cut down the amount of data shown to a reasonable level.

```
{
  "version": 8,
  "name": "point_example",
  "sprite
↪": "http://localhost:8080/geoserver/styles/sprites"
  "layers": [
    {
      "id": "point_example",
      "type": "symbol",
      "source-layer": "ne:populated_places",
      "filter": ["<", "SCALERANK", 1],
      "layout": {
        "icon-image": "grey_square16",
      }
    }
  ]
}
```

4. Resulting in a considerably cleaner image:



5. Additional properties are available to control an icon's presentation:

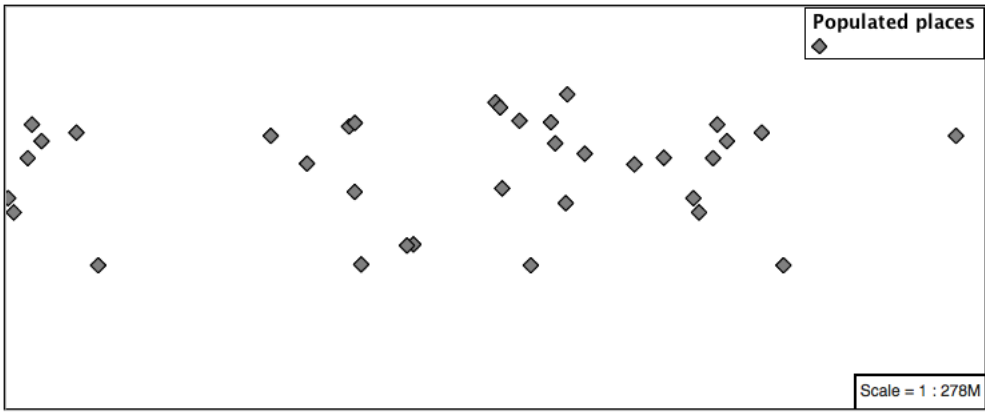
   The **icon-size** property is used to control symbol size.

   The **icon-rotate** property controls orientation, accepting input in degrees.

Trying these two settings together:

```
{
  "version": 8,
  "name": "point_example",
  "sprite
↪": "http://localhost:8080/geoserver/styles/sprites"
  "layers": [
    {
      "id": "point_example",
      "type": "symbol",
      "source-layer": "ne:populated_places",
      "filter": ["<", "SCALERANK", 1],
      "layout": {
        "icon-image": "grey_square16",
        "icon-size": 0.75,
        "icon-rotate": 45
      }
    }
  ]
}
```

6. Results in each location being marked with a diamond:



Circle

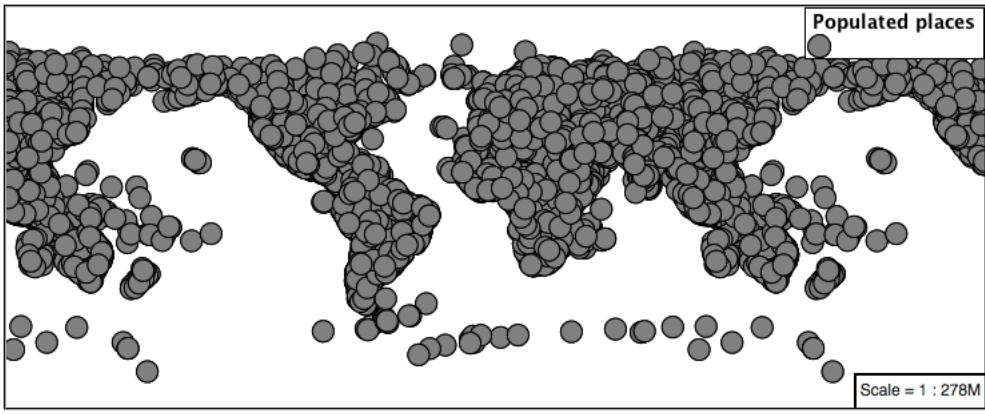
Another way of displaying point data is using the **circle** layer. Rather than rendering an icon from a preset sprite sheet, the circle layer lets us chose size and color for a simple circle.

1. Modify the style to render a grey circle using the **circle** layer:

```
{
  "version": 8,
  "name": "point_example",
  "layers": [
    {
      "id": "point_example",
      "type": "circle",
      "source-layer": "ne:populated_places",
      "paint": {
        "circle-color": "gray",
```

```
            "circle-radius": 8
            "circle-stroke-color": "black",
            "circle-stroke-width": 1
          }
        }
      ]
    }
```

2. And use the *Layer Preview* tab to preview the result.



Label

Labeling is now familiar from our experience with LineString and Polygons.

The **symbol** layer with the **label** property are used to to label Point Locations.

1. Replace `point_example` with the following:

```
{
  "version": 8,
  "name": "point_example",
  "layers": [
     {
      "id": "point_circle",
      "type": "circle",
      "source-layer": "ne:populated_places",
      "paint": {
        "circle-color": "gray",
        "circle-radius": 8
        "circle-stroke-color": "black",
        "circle-stroke-width": 1
      }
    },
    {
      "id": "point_label",
      "source-layer": "ne:populated_places",
      "filter": ["<", "SCALERANK", 1],
      "type": "symbol",
```

```
      "layout": {
        "text-field": "{NAME}"
      },
      "paint": {
        "text-color": "gray"
      }
    }
  ]
}
```

2. Confirm the result in `Map` preview.

3. Each label is drawn starting from the provided point - which is unfortunate as it assures each label will overlap with the symbol used. To fix this limitation we will make use of the MBStyle controls for label placement:

   text-anchor provides a value expressing how a label is aligned with respect to the starting label position.

   text-translate is used to provide an initial displacement using and x and y offset. For points this offset is recommended to adjust the label position away for the area used by the symbol.

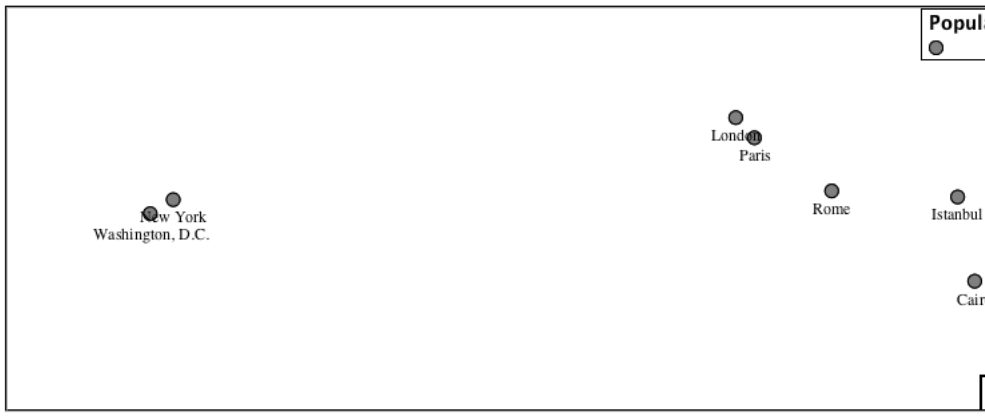
   Note: The property **text-anchor** defines an anchor position relative to the bounding box formed by the resulting label. This anchor position is snapped to the label position generated by the point location and displacement offset.

4. Using these two facilities together we can center our labels below the symbol, taking care that the displacement used provides an offset just outside the area required for the symbol size.

```
{
  "version": 8,
  "name": "point_example",
  "layers": [
    {
      "id": "point_circle",
      "type": "circle",
      "source-layer": "ne:populated_places",
```

```
      "paint": {
        "circle-color": "gray",
        "circle-radius": 8
        "circle-stroke-color": "black",
        "circle-stroke-width": 1
      }
    },
    {
      "id": "point_label",
      "source-layer": "ne:populated_places",
      "filter": ["<", "SCALERANK", 1],
      "type": "symbol",
      "layout": {
        "text-field": "{NAME}",
        "text-anchor": "top"
      },
      "paint": {
        "text-color": "black",
        "text-translate": [0, 12]
      }
    }
  ]
}
```

5. Each label is now placed under the mark.



6. One remaining issue is the overlap between labels and symbols.

   MBStyle provides various parameters to control label rendering and conflict resolution, preventing labels from overlapping any symbols.

   icon-allow-overlap and **text-allow-overlap** allows the rendering engine to draw the indicated symbol atop previous labels and icons.

   icon-ignore-placement and **text-ignore-placement** allows the rendering engine to draw labels and icons over top of the indicated symbol.

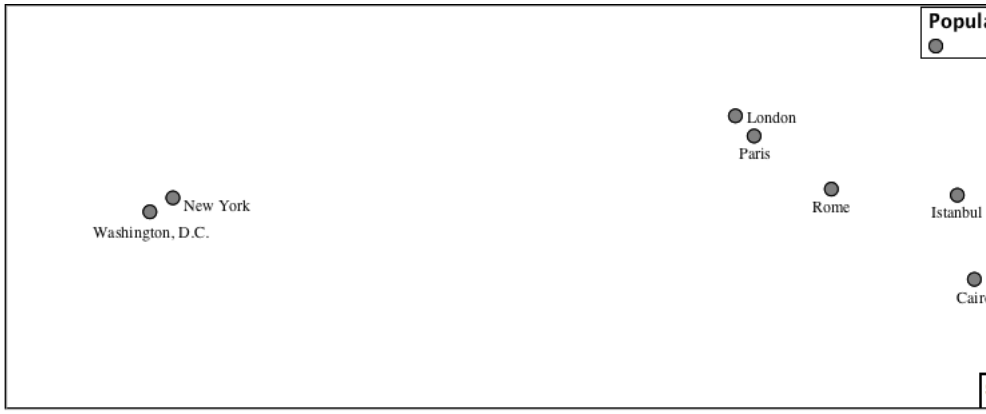
   icon-padding and **text-padding** tells the rendering engine to provide a minimum distance between the icons and text on the map, ensuring they do not overlap with other labels or icons.

   The **-allow-overlap** and **-ignore-placement** parameters are false by default, which is the behavior we want. Update our example to use

text-padding:

```
{
  "version": 8,
  "name": "point_example",
  "layers": [
    {
      "id": "point_circle",
      "type": "circle",
      "source-layer": "ne:populated_places",
      "paint": {
        "circle-color": "gray",
        "circle-radius": 8
        "circle-stroke-color": "black",
        "circle-stroke-width": 1
      }
    },
    {
      "id": "point_label",
      "source-layer": "ne:populated_places",
      "filter": ["<", "SCALERANK", 1],
      "type": "symbol",
      "layout": {
        "text-field": "{NAME}",
        "text-anchor": "top",
        "text-padding": 2
      },
      "paint": {
        "text-color": "black",
        "text-translate": [0, 12]
      }
    }
  ]
}
```

7. Resulting in a considerably cleaner image:



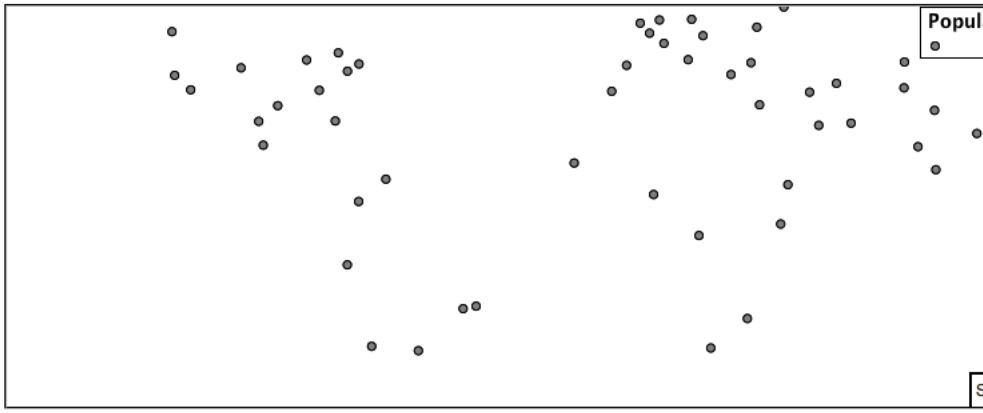
Dynamic Styling

1. We will quickly use **minzoom** and **maxzoom** to select content based on SCALERANK selectors.

```
{
  "version": 8,
  "name": "point_example",
  "layers": [
    {
      "id": "point_7",
      "type": "circle",
      "source-layer": "ne:populated_places",
      "filter": ["<", "SCALERANK", 7],
      "minzoom": 6,
      "maxzoom": 7,
      "paint": {
        "circle-color": "gray",
        "circle-radius": 8
        "circle-stroke-color": "black",
        "circle-stroke-width": 1
      }
    },
    {
      "id": "point_5",
      "type": "circle",
      "source-layer": "ne:populated_places",
      "filter": ["<", "SCALERANK", 5],
      "minzoom": 5,
      "maxzoom": 6,
      "paint": {
        "circle-color": "gray",
        "circle-radius": 8
        "circle-stroke-color": "black",
        "circle-stroke-width": 1
      }
    },
    {
      "id": "point_4",
      "type": "circle",
      "source-layer": "ne:populated_places",
      "filter": ["<", "SCALERANK", 4],
      "minzoom": 4,
      "maxzoom": 5,
      "paint": {
        "circle-color": "gray",
        "circle-radius": 8
        "circle-stroke-color": "black",
        "circle-stroke-width": 1
      }
    },
    {
      "id": "point_3",
      "type": "circle",
      "source-layer": "ne:populated_places",
      "filter": ["<", "SCALERANK", 3],
      "minzoom": 3,
      "maxzoom": 4,
      "paint": {
        "circle-color": "gray",
        "circle-radius": 8
        "circle-stroke-color": "black",
        "circle-stroke-width": 1
```

```
        }
      },
      {
        "id": "point_2",
        "type": "circle",
        "source-layer": "ne:populated_places",
        "filter": ["<", "SCALERANK", 2],
        "minzoom": 2,
        "maxzoom": 3,
        "paint": {
          "circle-color": "gray",
          "circle-radius": 8
          "circle-stroke-color": "black",
          "circle-stroke-width": 1
        }
      },
      {
        "id": "point_1",
        "type": "circle",
        "source-layer": "ne:populated_places",
        "filter": ["<", "SCALERANK", 1],
        "maxzoom": 2,
        "paint": {
          "circle-color": "gray",
          "circle-radius": 8
          "circle-stroke-color": "black",
          "circle-stroke-width": 1
        }
      },
      {
        "id": "point_0",
        "type": "circle",
        "source-layer": "ne:populated_places",
        "minzoom": 7,
        "paint": {
          "circle-color": "gray",
          "circle-radius": 8
          "circle-stroke-color": "black",
          "circle-stroke-width": 1
        }
      }
    ]
  }
```

2. Click *Submit* to update the *Map* after each step.



3. To add labeling we can add a symbol layer for each of the existing circle layers.

```
{
   "version": 8,
   "name": "point_example",
   "layers": [
     {
       "id": "point_7",
       "type": "circle",
       "source-layer": "ne:populated_places",
       "filter": ["<", "SCALERANK", 7],
       "minzoom": 6,
       "maxzoom": 7,
       "paint": {
         "circle-color": "gray",
         "circle-radius": 8
         "circle-stroke-color": "black",
         "circle-stroke-width": 1
       }
     },
     {
       "id": "point_7_text",
       "type": "symbol",
       "source-layer": "ne:populated_places",
       "filter": ["<", "SCALERANK", 7],
       "minzoom": 6,
       "maxzoom": 7,
       "layout": {
         "text-field": "{NAME}",
         "text-font": ["Arial"],
         "text-size": 10
       },
       "paint": {
         "text-color": "black"
       }
     },
     {
       "id": "point_5",
       "type": "circle",
       "source-layer": "ne:populated_places",
       "filter": ["<", "SCALERANK", 5],
       "minzoom": 5,
       "maxzoom": 6,
```

```
      "paint": {
        "circle-color": "gray",
        "circle-radius": 8
        "circle-stroke-color": "black",
        "circle-stroke-width": 1
      }
    },
    {
      "id": "point_5_text",
      "type": "symbol",
      "source-layer": "ne:populated_places",
      "filter": ["<", "SCALERANK", 5],
      "minzoom": 5,
      "maxzoom": 6,
      "layout": {
        "text-field": "{NAME}",
        "text-font": ["Arial"],
        "text-size": 10
      },
      "paint": {
        "text-color": "black"
      }
    },
    {
      "id": "point_4",
      "type": "circle",
      "source-layer": "ne:populated_places",
      "filter": ["<", "SCALERANK", 4],
      "minzoom": 4,
      "maxzoom": 5,
      "paint": {
        "circle-color": "gray",
        "circle-radius": 8
        "circle-stroke-color": "black",
        "circle-stroke-width": 1
      }
    },
    {
      "id": "point_4_text",
      "type": "symbol",
      "source-layer": "ne:populated_places",
      "filter": ["<", "SCALERANK", 4],
      "minzoom": 4,
      "maxzoom": 5,
      "layout": {
        "text-field": "{NAME}",
        "text-font": ["Arial"],
        "text-size": 10
      },
      "paint": {
        "text-color": "black"
      }
    },
    {
      "id": "point_3",
      "type": "circle",
      "source-layer": "ne:populated_places",
      "filter": ["<", "SCALERANK", 3],
```

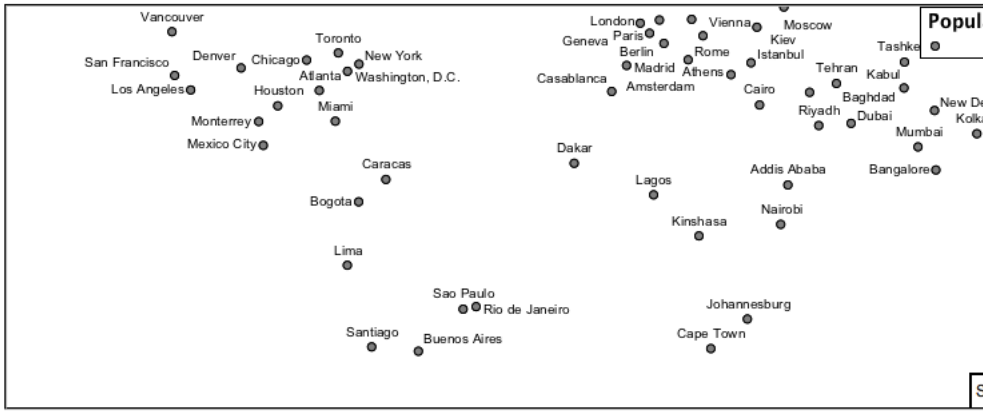
```
      "minzoom": 3,
      "maxzoom": 4,
      "paint": {
        "circle-color": "gray",
        "circle-radius": 8
        "circle-stroke-color": "black",
        "circle-stroke-width": 1
      }
    },
    {
      "id": "point_3_text",
      "type": "symbol",
      "source-layer": "ne:populated_places",
      "filter": ["<", "SCALERANK", 3],
      "minzoom": 3,
      "maxzoom": 4,
      "layout": {
        "text-field": "{NAME}",
        "text-font": ["Arial"],
        "text-size": 10
      },
      "paint": {
        "text-color": "black"
      }
    },
    {
      "id": "point_2",
      "type": "circle",
      "source-layer": "ne:populated_places",
      "filter": ["<", "SCALERANK", 2],
      "minzoom": 2,
      "maxzoom": 3,
      "paint": {
        "circle-color": "gray",
        "circle-radius": 8
        "circle-stroke-color": "black",
        "circle-stroke-width": 1
      }
    },
    {
      "id": "point_2_text",
      "type": "symbol",
      "source-layer": "ne:populated_places",
      "filter": ["<", "SCALERANK", 2],
      "minzoom": 2,
      "maxzoom": 3,
      "layout": {
        "text-field": "{NAME}",
        "text-font": ["Arial"],
        "text-size": 10
      },
      "paint": {
        "text-color": "black"
      }
    },
    {
      "id": "point_1",
      "type": "circle",
```

```
      "source-layer": "ne:populated_places",
      "filter": ["<", "SCALERANK", 1],
      "maxzoom": 2,
      "paint": {
        "circle-color": "gray",
        "circle-radius": 8
        "circle-stroke-color": "black",
        "circle-stroke-width": 1
      }
    },
    {
      "id": "point_1_text",
      "type": "symbol",
      "source-layer": "ne:populated_places",
      "filter": ["<", "SCALERANK", 1],
      "maxzoom": 2,
      "layout": {
        "text-field": "{NAME}",
        "text-font": ["Arial"],
        "text-size": 10
      },
      "paint": {
        "text-color": "black"
      }
    },
    {
      "id": "point_0",
      "type": "circle",
      "source-layer": "ne:populated_places",
      "minzoom": 7,
      "paint": {
        "circle-color": "gray",
        "circle-radius": 8
        "circle-stroke-color": "black",
        "circle-stroke-width": 1
      }
    },
    {
      "id": "point_0_text",
      "type": "symbol",
      "source-layer": "ne:populated_places",
      "minzoom": 7,
      "layout": {
        "text-field": "{NAME}",
        "text-font": ["Arial"],
        "text-size": 10
      },
      "paint": {
        "text-color": "black"
      }
    }
  ]
}
```

4. We will use **text-offset** to position the label above each symbol, and **text-padding** to give some extra space around our labels.

   Add the following line to each layer:

```
{
  "id": "point_example",
  "type": "symbol",
  "source-layer": "ne:populated_places",
  "minzoom": 7,
  "layout": {
    "text-field": "{NAME}",
    "text-font": ["Arial"],
    "text-size": 10,
    "text-padding": 2
  },
  "paint": {
    "text-color": "black",
    "text-translate": [0, -12]
  }
}
```



5. Now that we have clearly labeled our cities, zoom into an area you are familiar with and we can look at changing symbology on a case-by-case basis.

   We have used expressions previous to generate an appropriate label. Expressions can also be used for many other property settings.

   The `ne:populated_places` layer provides several attributes

specifically to make styling easier:

- **SCALERANK**: we have already used this attribute to control the level of detail displayed
- **FEATURECLA**: used to indicate different types of cities. We will check for `Admin-0 capital` cities.

The first thing we will do is calculate the point **size** using a quick expression:

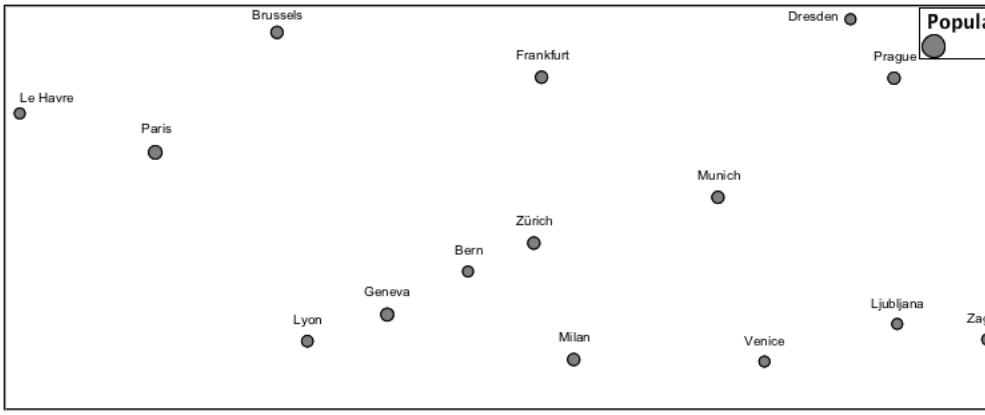
```
{
  "property": "SCALERANK",
  "type": "exponential",
  "stops": [
    [0, 4.5],
    [10, 2.5]
  ]
},
```

This expression should result in sizes between 5 and 9 and will need to be applied to both point **size** and label **displacement**.

```
{
  "id": "point_0",
  "type": "circle",
  "source-layer": "ne:populated_places",
  "minzoom": 7,
  "paint": {
    "circle-color": "gray",
    "circle-radius": {
      "property": "SCALERANK",
      "type": "exponential",
      "stops": [
        [0, 4.5],
        [10, 2.5]
      ]
    },
    "circle-stroke-color": "black",
    "circle-stroke-width": 1
  }
}
```

```
{
  "id": "point_0_text",
  "type": "symbol",
  "source-layer": "ne:populated_places",
  "minzoom": 7,
  "layout": {
    "text-field": "{NAME}",
    "text-font": ["Arial"],
    "text-size": 10,
    "text-padding": 2
  },
  "paint": {
    "text-color": "black",
    "text-translate": {
      "property": "SCALERANK",
      "type": "exponential",
```

```
        "stops": [
          [0, [0, -8]],
          [10, [0, -6]]
        ]
      },
    }
  }
```



6. Next we can use `FEATURECLA` to check for capital cities.

   Adding a selector for capital cities at the top of the **rules** list:

```
    {
      "id": "point_capital",
      "type": "symbol",
      "source-layer": "ne:populated_places",
      "filter": ["all",["<", "SCALERANK
    ↪", 2], ["==", "FEATURECLA", "Admin-0 capital"]]
      "minzoom": 2,
      "layout": {
        "icon-image": "star",
        "text-field": "{NAME}",
        "text-font": ["Arial"],
        "text-size": 10,
        "text-padding": 2
      },
      "paint": {
        "text-color": "black",
        "text-translate": [0, -12]
      }
    }
```

Also add the spritesheet url to the top of the style if it is not present:

```
    {
      "version": 8,
      "name": "point_example",
      "sprite
    ↪": "http://localhost:8080/geoserver/styles/sprites",
    }
```

And updating the populated places selectors to ignore capital cities:

```
{
  "id": "point_7",
  "type": "circle",
  "source-layer": "ne:populated_places",
  "filter": ["all", ["<", "SCALERANK
↪", 7], ["!=", "FEATURECLA", "Admin-0 capital"]],
  "minzoom": 6,
  "maxzoom": 7,
  "paint": {
    "circle-color": "gray",
    "circle-radius": {
      "property": "SCALERANK",
      "type": "exponential",
      "stops": [
        [0, 4.5],
        [10, 2.5]
      ]
    },
    "circle-stroke-color": "black",
    "circle-stroke-width": 1
  }
}
```

```
{
  "id": "point_7_text",
  "type": "symbol",
  "source-layer": "ne:populated_places",
  "filter": ["all", ["<", "SCALERANK
↪", 7], ["!=", "FEATURECLA", "Admin-0 capital"]],
  "minzoom": 6,
  "maxzoom": 7,
  "layout": {
    "text-field": "{NAME}",
    "text-font": ["Arial"],
    "text-size": 10,
    "text-padding": 2
  },
  "paint": {
    "text-color": "black",
    "text-translate": {
      "property": "SCALERANK",
      "type": "exponential",
      "stops": [
        [0, [0, -8]],
        [10, [0, -6]]
      ]
    }
  }
}
```

```
{
  "id": "point_5",
  "type": "circle",
  "source-layer": "ne:populated_places",
  "filter": ["all", ["<", "SCALERANK
↪", 5], ["!=", "FEATURECLA", "Admin-0 capital"]],
  "minzoom": 5,
```

```
      "maxzoom": 6,
      "paint": {
        "circle-color": "gray",
        "circle-radius": {
          "property": "SCALERANK",
          "type": "exponential",
          "stops": [
            [0, 4.5],
            [10, 2.5]
          ]
        },
        "circle-stroke-color": "black",
        "circle-stroke-width": 1
      }
    }
```

```
    {
      "id": "point_5_text",
      "type": "symbol",
      "source-layer": "ne:populated_places",
      "filter": ["all", ["<", "SCALERANK
    ↪", 5], ["!=", "FEATURECLA", "Admin-0 capital"]],
      "minzoom": 5,
      "maxzoom": 6,
      "layout": {
        "text-field": "{NAME}",
        "text-font": ["Arial"],
        "text-size": 10,
        "text-padding": 2
      },
      "paint": {
        "text-color": "black",
        "text-translate": {
          "property": "SCALERANK",
          "type": "exponential",
          "stops": [
            [0, [0, -8]],
            [10, [0, -6]]
          ]
        }
      }
    }
```

```
    {
      "id": "point_4",
      "type": "circle",
      "source-layer": "ne:populated_places",
      "filter": ["all", ["<", "SCALERANK
    ↪", 4], ["!=", "FEATURECLA", "Admin-0 capital"]],
      "minzoom": 4,
      "maxzoom": 5,
      "paint": {
        "circle-color": "gray",
        "circle-radius": {
          "property": "SCALERANK",
          "type": "exponential",
          "stops": [
```

```
            [0, 4.5],
            [10, 2.5]
          ]
        },
        "circle-stroke-color": "black",
        "circle-stroke-width": 1
      }
    }
```

```
    {
      "id": "point_4_text",
      "type": "symbol",
      "source-layer": "ne:populated_places",
      "filter": ["all", ["<", "SCALERANK
    ↪", 4], ["!=", "FEATURECLA", "Admin-0 capital"]],
      "minzoom": 4,
      "maxzoom": 5,
      "layout": {
        "text-field": "{NAME}",
        "text-font": ["Arial"],
        "text-size": 10,
        "text-padding": 2
      },
      "paint": {
        "text-color": "black",
        "text-translate": {
          "property": "SCALERANK",
          "type": "exponential",
          "stops": [
            [0, [0, -8]],
            [10, [0, -6]]
          ]
        }
      }
    }
```

```
    {
      "id": "point_3",
      "type": "circle",
      "source-layer": "ne:populated_places",
      "filter": ["all", ["<", "SCALERANK
    ↪", 3], ["!=", "FEATURECLA", "Admin-0 capital"]],
      "minzoom": 3,
      "maxzoom": 4,
      "paint": {
        "circle-color": "gray",
        "circle-radius": 8
        "circle-stroke-color": "black",
        "circle-stroke-width": 1
      }
    }
```

```
    {
      "id": "point_3_text",
      "type": "symbol",
      "source-layer": "ne:populated_places",
      "filter": ["all", ["<", "SCALERANK
    ↪", 3], ["!=", "FEATURECLA", "Admin-0 capital"]],
```

```
      "minzoom": 3,
      "maxzoom": 4,
      "layout": {
        "text-field": "{NAME}",
        "text-font": ["Arial"],
        "text-size": 10,
        "text-padding": 2
      },
      "paint": {
        "text-color": "black",
        "text-translate": {
          "property": "SCALERANK",
          "type": "exponential",
          "stops": [
            [0, [0, -8]],
            [10, [0, -6]]
          ]
        }
      }
    }
```

```
    {
      "id": "point_2",
      "type": "circle",
      "source-layer": "ne:populated_places",
      "filter": ["all", ["<", "SCALERANK
    ↪", 2], ["!=", "FEATURECLA", "Admin-0 capital"]],
      "minzoom": 2,
      "maxzoom": 3,
      "paint": {
        "circle-color": "gray",
        "circle-radius": {
          "property": "SCALERANK",
          "type": "exponential",
          "stops": [
            [0, 4.5],
            [10, 2.5]
          ]
        },
        "circle-stroke-color": "black",
        "circle-stroke-width": 1
      }
    }
```

```
    {
      "id": "point_2_text",
      "type": "symbol",
      "source-layer": "ne:populated_places",
      "filter": ["all", ["<", "SCALERANK
    ↪", 2], ["!=", "FEATURECLA", "Admin-0 capital"]],
      "minzoom": 2,
      "maxzoom": 3,
      "layout": {
        "text-field": "{NAME}",
        "text-font": ["Arial"],
        "text-size": 10,
        "text-padding": 2
```

```
      },
      "paint": {
        "text-color": "black",
        "text-translate": {
          "property": "SCALERANK",
          "type": "exponential",
          "stops": [
            [0, [0, -8]],
            [10, [0, -6]]
          ]
        }
      }
    }
```

```
    {
      "id": "point_1",
      "type": "circle",
      "source-layer": "ne:populated_places",
      "filter": ["<", "SCALERANK", 1],
      "maxzoom": 2,
      "paint": {
        "circle-color": "gray",
        "circle-radius": {
          "property": "SCALERANK",
          "type": "exponential",
          "stops": [
            [0, 4.5],
            [10, 2.5]
          ]
        },
        "circle-stroke-color": "black",
        "circle-stroke-width": 1
      }
    }
```

```
    {
      "id": "point_1_text",
      "type": "symbol",
      "source-layer": "ne:populated_places",
      "filter": ["<", "SCALERANK", 1],
      "maxzoom": 2,
      "layout": {
        "text-field": "{NAME}",
        "text-font": ["Arial"],
        "text-size": 10,
        "text-padding": 2
      },
      "paint": {
        "text-color": "black",
        "text-translate": {
          "property": "SCALERANK",
          "type": "exponential",
          "stops": [
            [0, [0, -8]],
            [10, [0, -6]]
          ]
        }
```

```
    }
  }
```

```
  {
    "id": "point_0",
    "type": "circle",
    "source-layer": "ne:populated_places",
    "filter": ["!=", "FEATURECLA", "Admin-0 capital"],
    "minzoom": 7,
    "paint": {
      "circle-color": "gray",
      "circle-radius": {
        "property": "SCALERANK",
        "type": "exponential",
        "stops": [
          [0, 4.5],
          [10, 2.5]
        ]
      },
      "circle-stroke-color": "black",
      "circle-stroke-width": 1
    }
  }
```

```
  {
    "id": "point_0_text",
    "type": "symbol",
    "source-layer": "ne:populated_places",
    "filter": ["!=", "FEATURECLA", "Admin-0 capital"],
    "minzoom": 7,
    "layout": {
      "text-field": "{NAME}",
      "text-font": ["Arial"],
      "text-size": 10,
      "text-padding": 2
    },
    "paint": {
      "text-color": "black",
      "text-translate": {
        "property": "SCALERANK",
        "type": "exponential",
        "stops": [
          [0, [0, -8]],
          [10, [0, -6]]
        ]
      }
    }
  }
```

7. If you would like to check your work the final file is here: `point_example.mbstyle`

Bonus

Challenge Geometry Location

1. The **mark** property can be used to render any geometry content.
2. **Challenge:** Try this yourself by rendering a polygon layer using a **mark** property.

Note: Answer *discussed* at the end of the workbook.

Explore Dynamic Symbolization

1. We went to a lot of work to set up selectors to choose between star and circle for capital cities.

   This approach is straightforward when applied in isolation:

```
{
  "version": 8,
  "name": "point_example",
  "sprite
↪": "http://localhost:8080/geoserver/styles/sprites",
  "layers": [
    {
      "id": "point_capital",
      "type": "symbol",
      "source-layer": "ne:populated_places",
      "filter": ["==", "FEATURECLA", "Admin-0 capital"]
      "minzoom": 2,
      "layout": {
        "icon-image": "star",
      }
    },
    {
      "id": "point_0",
```

```
      "type": "circle",
      "source-layer": "ne:populated_places",
      "filter": ["!=", "FEATURECLA", "Admin-0 capital"],
      "minzoom": 7,
      "paint": {
        "circle-color": "gray",
        "circle-radius": 4,
        "circle-stroke-color": "black",
        "circle-stroke-width": 1
      }
    }
  ]
}
```

When combined with checking another attribute, or checking @scale as in our example, this approach can quickly lead to many rules which can be difficult to keep straight.

2. Taking a closer look, `icon-image` is expressed using a string:

```
{
  "id": "point_capital",
  "type": "symbol",
  "source-layer": "ne:populated_places",
  "filter": ["==", "FEATURECLA", "Admin-0 capital"]
  "minzoom": 2,
  "layout": {
    "icon-image": "star",
  }
}
```

Which is represented in SLD as:

```
<sld:PointSymbolizer
 uom="http://www.opengeospatial.org/se/units/pixel">
  <sld:Graphic>
    <sld:ExternalGraphic>
      <sld:OnlineResource xmlns:xlink="http://www.
 w3.org/1999/xlink" xlink:type="simple" xlink:href=
 "http://localhost:8080/geoserver/styles/sprites#icon=
 ${strURLEncode('star')}&size=${strURLEncode(1.0)}"/>
      <sld:Format>mbsprite</sld:Format>
    </sld:ExternalGraphic>
  </sld:Graphic>
</sld:PointSymbolizer>
```

3. MBStyle provides an opportunity for dynamic symbolization.

   This is accomplished by using a function for the value of the **icon-image**:

```
{
  "version": 8,
  "name": "point_example",
  "sprite
 ": "http://localhost:8080/geoserver/styles/sprites",
  "layers": [
```

```
                {
                  "id": "point_capital",
                  "type": "symbol",
                  "source-layer": "ne:populated_places",
                  "layout": {
                    "icon-image": {
                      "type": "categorical",
                      "property": "FEATURECLA",
                      "default": "grey_circle",
                      "stops": [
                        ["Admin-0 capital", "star"]
                      ]
                    }
                  }
                }
              ]
            }
```

Which is represented in SLD as:

```
<sld:PointSymbolizer␣
↪uom="http://www.opengeospatial.org/se/units/pixel">
  <sld:Graphic>
    <sld:ExternalGraphic>
      <sld:OnlineResource xmlns:xlink="http://www.
↪w3.org/1999/xlink" xlink:type="simple" xlink:href=
↪"http://localhost:8080/geoserver/styles/sprites
↪#icon=${strURLEncode(DefaultIfNull(Recode(FEATURECLA,
↪'Admin-0 capital',
↪'star'),'grey_circle'))}&size=${strURLEncode(1.0)}"/>
      <sld:Format>mbsprite</sld:Format>
    </sld:ExternalGraphic>
  </sld:Graphic>
</sld:PointSymbolizer>
```

4. **Challenge:** Use this approach to rewrite the *Dynamic Styling* example.

Note: Answer *provided* at the end of the workbook.

Rasters

Finally we will look at using MBStyle styling for the portrayal of raster data.

Fig. 6.315: Raster Symbology

Review of raster symbology:

- Raster data is **Grid Coverage** where values have been recorded in a regular array. In OGC terms a **Coverage** can be used to look up a value or measurement for each location.
- When queried with a "sample" location:

  - A grid coverage can determine the appropriate array location and retrieve a value. Different techniques may be used interpolate an appropriate value from several measurements (higher quality) or directly return the "nearest neighbor" (faster).
  - A vector coverages would use a point-in-polygon check and return an appropriate attribute value.
  - A scientific model can calculate a value for each sample location
- Many raster formats organize information into bands of content. Values recorded in these bands and may be mapped into colors for display (a process similar to theming an attribute for vector data).

  For imagery the raster data is already formed into red, green and blue bands for display.
- As raster data has no inherent shape, the format is responsible for describing the orientation and location of the grid used to record measurements.

  These raster examples use a digital elevation model consisting of a single band of height measurements. The imagery examples use an RGB image that has been hand coloured for use as a base map.

  Since MBStyle is primarily intended for client-side styling, it doesn't have much ability to style raster data when compared with SLD, so this section will be much shorter than the equivalent raster sections for CSS and YSLD.

  Reference:

- *MBStyle Reference*
- MapBox Style Spec Raster Layer
- *Point* (User Manual | SLD Reference)

The exercise makes use of the `usgs:dem` and `ne:ne1` layers.

Image

The **raster** layer controls the display of raster data.

1. Navigate to the **Styles** page.
2. Click *Add a new style* and choose the following:

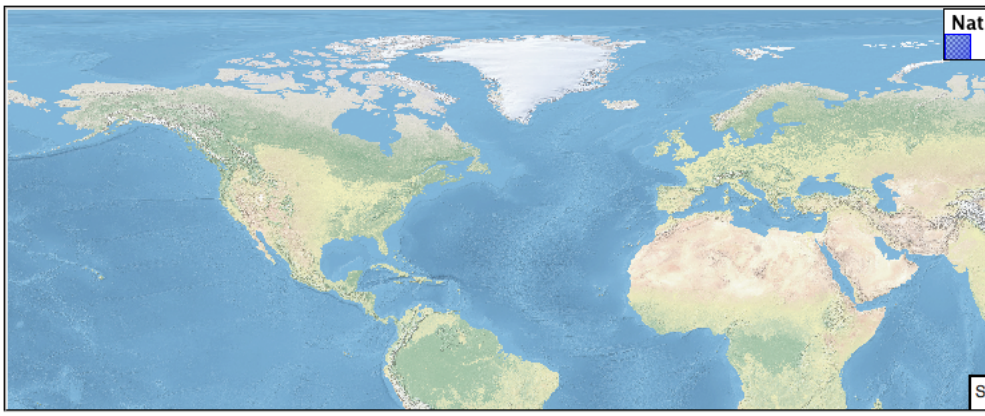
Name:	`image_example`
Workspace:	`No workspace`
Format:	`MBStyle`

3. Replace the initial MBStyle definition with:

```
{
  "name": "image_example",
  "version": 8,
  "layers": [
    {
      "id": "image_example",
```

```
      "type": "raster",
      "source-layer": "ne:ne1",
      "paint": {
        "raster-opacity": 1
      }
    }
  ]
}
```

4. And use the *Layer Preview* tab to preview the result.



DEM

A digital elevation model is an example of raster data made up of measurements, rather than color information.

The `usgs:dem` layer used used for this exercise:

1. Return to the the **Styles** page.
2. Click *Add a new style* and choose the following:

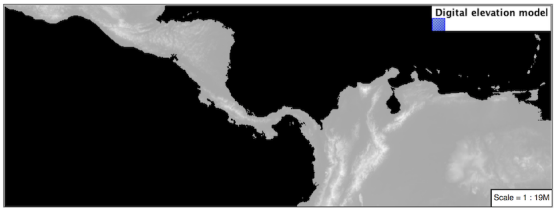
Name:	`raster_example`
Workspace:	`No workspace`
Format:	`MBStyle`

3. The rendering engine will select our single band of raster content, and do its best to map these values into a grayscale image. Replace the content of the style with:

```
{
  "name": "raster_example",
  "version": 8,
  "layers": [
    {
      "id": "raster_example",
      "type": "raster",
      "source-layer": "usgs:dem",
      "paint": {
        "raster-opacity": 1
      }
```

```
                }
            ]
        }
```

4. Use the *Layer Preview* tab to preview the result. The range produced in this case from the highest and lowest values.



Bonus

Challenge Raster Opacity

1. There is a quick way to make raster data transparent, raster **opacity** property works in the same fashion as with vector data. The raster as a whole will be drawn partially transparent allow content from other layers to provide context.
2. **Challenge:** Can you think of an example where this would be useful?

Note: Discussion *provided* at the end of the workbook.

MBStyle Workbook Conclusion

We hope you have enjoyed this styling workshop.

Additional resources:

- *MBStyle Extension*
- *MBStyle Reference*

MBStyle Tips and Tricks

MBStyle Workshop Answer Key

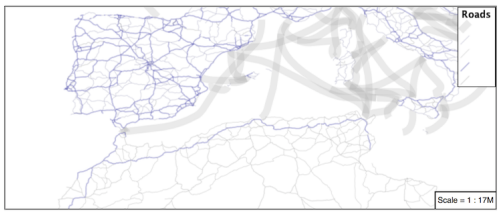
The following questions were listed through out the workshop as an opportunity to explore the material in greater depth. Please do your best to consider the questions in detail prior to checking here for the answer. Questions are provided to teach valuable skills, such as a chance to understand how feature type styles are used to control z-order, or where to locate information in the user manual.

Note: Coming Soon

Classification

Answer for *Challenge Classification*:

1. **Challenge:** Create a new style adjust road appearance based on **type**.



Hint: The available values are 'Major Highway','Secondary Highway','Road' and 'Unknown'.

2. Here is an example:

```
{
 "version": 8,
 "name": "line_example",
 "layers": [
   {
     "id": "line_hwy1",
     "source-layer": "ne:roads",
     "type": "line",
     "filter": ["==", "type", "Major Highway"],
     "paint": {
       "line-color": "#000088",
       "line-width": 1.25,
       "line-opacity": 0.25
     }
   },
   {
     "id": "line_hwy2",
     "source-layer": "ne:roads",
     "type": "line",
     "filter": ["==", "type", "Secondary Highway"],
     "paint": {
       "line-color": "#8888AA",
       "line-width": 0.75,
       "line-opacity": 0.25
     }
   },
   {
     "id": "line_road",
     "source-layer": "ne:roads",
     "type": "line",
     "filter": ["==", "type", "Road"],
     "paint": {
       "line-color": "#888888",
       "line-width": 0.75,
```

```
      "line-opacity": 0.25
    }
  },
  {
    "id": "line_unk",
    "source-layer": "ne:roads",
    "type": "line",
    "filter": ["==", "type", "Unknown"],
    "paint": {
      "line-color": "#888888",
      "line-width": 0.5,
      "line-opacity": 0.25
    }
  }
 ]
}
```

One Rule Classification

Answer for *Challenge One Rule Classification*:

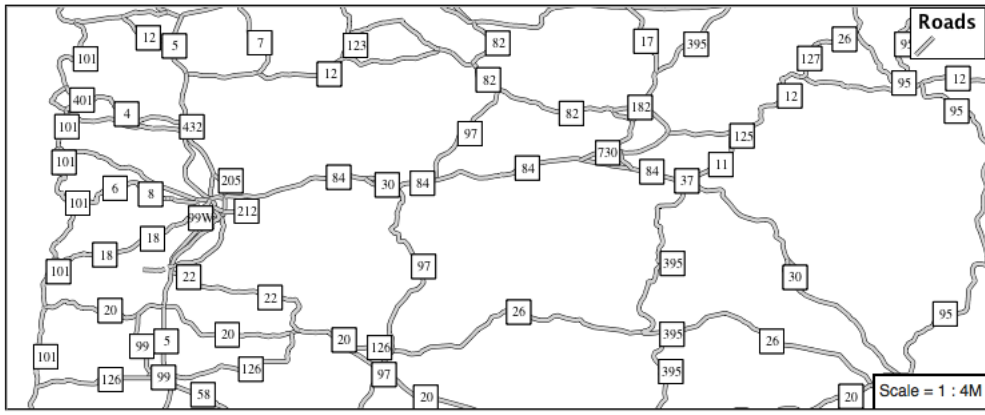
1. **Challenge:** Create a new style and classify the roads based on their scale rank using expressions in a single rule instead of multiple rules with filters.
2. This exercise requires looking up information in the MBstyle user guide.

- The Mapbox Style specification functions provides details.

Label Shields

Answer for *Challenge Label Shields*:

1. *Challenge:* Have a look at the documentation for putting a graphic on a text symbolizer in SLD and reproduce this technique in MBStyle.



2. The use of a label shield is a vendor specific capability of the GeoServer rendering engine. The tricky part of this exercise is finding which symbol layout parameters give the desired behavior,

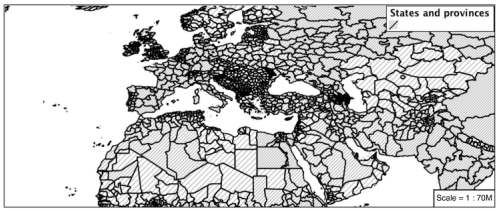
mainly **icon-text-fit** but also the various placement and overlap parameters to allow the text to be drawn atop the labels (see symbol layer).

```
{
  "version": 8,
  "name": "line_example",
  "sprite
↪": "http://localhost:8080/geoserver/styles/sprites",
  "layers": [
    {
      "id": "line_casing",
      "source-layer": "ne:roads",
      "type": "line",
      "paint": {
        "line-color": "#000000",
        "line-width": 3,
      }
    },
    {
      "id": "line_inner",
      "source-layer": "ne:roads",
      "type": "line",
      "paint": {
        "line-color": "#D3D3D3",
        "line-width": 2,
      }
    },
    {
      "id": "label",
      "source-layer": "ne:roads",
      "type": "symbol",
      "layout": {
        "icon-image": "white_square16",
        "icon-text-fit": "width",
        "icon-text-fit-padding": [2, 2, 2, 2],
        "text-field": "{name}",
        "text-font": ["Ariel"],
        "text-font-size": 10,
        "text-ignore-placement": true,
        "text-allow-overlap": true,
        "icon-ignore-placement": true,
        "icon-allow-overlap": true,
        "symbol-placement": "line",
        "symbol-spacing": 0

      }
      "paint": {
        "text-color": "black"
      }
    }
  ]
}
```

Interval

Answer for *Explore Interval*:

1. An exciting use of the GeoServer **fill-pattern** symbols is theming by changing the pattern used.
2. **Explore:** Use the **interval** function to theme by **datarank**.



Example:

```
{
 "version": 8,
 "name": "polygon_example",
 "sprite
↪": "http://localhost:8080/geoserver/styles/sprites",
 "layers": [
   {
     "id": "polygon",
     "source-layer": "ne:states_provinces_shp",
     "type": "fill",
     "paint": {
       "fill-pattern": {
         "property": "datarank",
         "type": "interval",
         "stops": [
           [4, "grey_diag8"],
           [6, "grey_diag16"]
         ]
       }
     }
   }
 ]
}
```

Halo

Answer for *Challenge Halo*:

1. The halo example used the fill color and opacity for a muted halo, while this improved readability it did not bring attention to our labels.

   A common design choice for emphasis is to outline the text in a contrasting color.

2. **Challenge:** Produce a map that uses a white halo around black text.

   Here is an example:

```
{
  "version": 8,
  "name": "polygon_example",
  "layers": [
    {
```

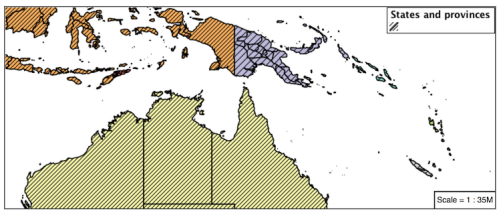
```
      "id": "polygon_fill",
      "source-layer": "ne:states_provinces_shp",
      "type": "fill",
      "paint": {
        "fill-color": "#7EB5D3",
        "fill-outline-color": "gray"
      }
    },
    {
      "id": "polygon_label",
      "source-layer": "ne:states_provinces_shp",
      "type": "symbol",
      "layout": {
        "text-field": "{name}",
        "text-anchor": "center"
        "text-max-width": 14,
        "text-font": ["Arial"]
      },
      "paint": {
        "text-color": "white",
        "text-halo-color": "black",
        "text-halo-width": 1

      }
    }
  ]
}
```

Theming using Multiple Attributes

Answer for *Challenge Theming using Multiple Attributes*:

1. A powerful tool is theming using multiple attributes. This is an important concept allowing map readers to perform "integration by eyeball" (detecting correlations between attribute values information).

2. **Challenge:** Combine the **mapcolor9** and **datarank** examples to reproduce the following map.



This should be a cut and paste using the `categorical` example, and `interval` examples already provided.

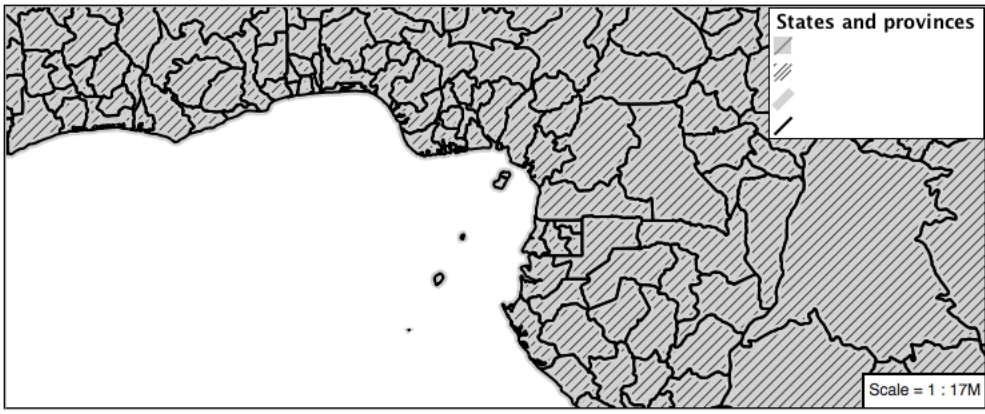
```
{
  "version": 8,
  "name": "polygon_example",
  "sprite
↪": "http://localhost:8080/geoserver/styles/sprites",
  "layers": [
```

```
                {
                  "id": "polygon",
                  "source-layer": "ne:states_provinces_shp",
                  "type": "fill",
                  "paint": {
                    "fill-color": {
                      "property": "mapcolor9",
                      "type": "categorical",
                      "stops": [
                        [1, "#8dd3c7"],
                        [2, "#ffffb3"],
                        [3, "#bebada"],
                        [4, "#fb8072"],
                        [5, "#80b1d3"],
                        [6, "#fdb462"],
                        [7, "#b3de69"],
                        [8, "#fccde5"],
                        [9, "#d9d9d9"]
                      ]
                    },
                    "fill-outline-color": "gray"
                  }
                },
                {
                  "id": "polygon",
                  "source-layer": "ne:states_provinces_shp",
                  "type": "fill",
                  "paint": {
                    "fill-pattern": {
                      "property": "datarank",
                      "type": "interval",
                      "stops": [
                        [4, "grey_diag8"],
                        [6, "grey_diag16"]
                      ]
                    }
                  }
                }
              ]
            }
```

Use of Z-Index

Answer for *Challenge Use of Z-Index*:

1. Using multiple **layers** to simulate line string casing. The resulting effect is similar to text halos - providing breathing space around complex line work allowing it to stand out.
2. **Challenge:** Use what you know of LineString rendering order to reproduce the following map:



This is much easier when using MBStyle, where z-order is controlled by layer.

```
{
   "version": 8,
   "name": "polygon_example",
   "sprite
↪": "http://localhost:8080/geoserver/styles/sprites",
   "layers": [
     {
       "id": "polygon_fill",
       "source-layer": "ne:states_provinces_shp",
       "type": "fill",
       "paint": {
         "fill-color": "lightgrey",
       }
     },
     {
       "id": "polygon_pattern",
       "source-layer": "ne:states_provinces_shp",
       "type": "fill",
       "paint": {
         "fill-pattern": "grey_diag16"
       }
     }
     {
       "id": "polygon_casing",
       "source-layer": "ne:states_provinces_shp",
       "type": "line",
       "paint": {
         "line-color": "lightgrey",
         "line-width": 6
       }
     },
     {
       "id": "polygon_outline",
       "source-layer": "ne:states_provinces_shp",
       "type": "line",
       "paint": {
         "line-color": "black",
         "line-width": 1.5
       }
     }
   ]
```

```
}
```

The structure of the legend graphic provides an indication on what is going on.

Geometry Location

Answer for *Challenge Geometry Location*:

1. The **symbol** layer can be used to render any geometry content.
2. **Challenge:** Try this yourself by rendering polygon data using a **symbol** layer.

   This can be done one of two ways:

- Changing the association of a polygon layer, such as `ne:states_provinces_shp` to point_example and using the layer preview page.
- Changing the *Layer Preview* tab to a polygon layer, such as `ne:states_provinces_shp`.

  The important thing to notice is that the centroid of each polygon is used as a point location.

Note: A layer in an MBStyle is not the same as a layer in GeoServer. A GeoServer layer is a raster or vector dataset that represents a collection of geographic features. A MBStyle layer is a block of styling information, similar to a SLD Symbolizer.

Dynamic Symbolization

Answer for *Explore Dynamic Symbolization*:

1. `icon-image` provides an opportunity for dynamic symbolization.

   This is accomplished by using a function for the value of `icon-image`:

```
{
  "version": 8,
  "name": "point_example",
  "sprite
↪": "http://localhost:8080/geoserver/styles/sprites",
  "layers": [
    {
      "id": "point_capital",
      "type": "symbol",
      "source-layer": "ne:populated_places",
      "layout": {
        "icon-image": {
          "type": "categorical",
          "property": "FEATURECLA",
          "default": "grey_circle",
          "stops": [
```

```
                                        ["Admin-0 capital", "star"]
                                    ]
                                }
                            }
                        }
                    ]
                }
```

2. **Challenge:** Use this approach to rewrite the *Dynamic Styling* example.

   Example available here `point_example.json`:

Raster Opacity

Discussion for *Challenge Raster Opacity*:

1. There is a quick way to make raster data transparent, raster **opacity** property works in the same fashion as with vector data. The raster as a whole will be drawn partially transparent allow content from other layers to provide context.
2. **Challenge:** Can you think of an example where this would be useful?

   This is difficult as raster data is usually provided for use as a basemap, with layers being drawn over top.

   The most obvious example here is the display of weather systems, or model output such as fire danger. By drawing the raster with some transparency, the landmass can be shown for context.

CHAPTER 7

Services

GeoServer serves data using standard protocols established by the Open Geospatial Consortium:

- The **Web Map Service** (WMS) supports requests for map images (and other formats) generated from geographical data.
- The **Web Feature Service** (WFS) supports requests for geographical feature data (with vector geometry and attributes).
- The **Web Coverage Service** (WCS) supports requests for coverage data (rasters).

These services are the primary way that GeoServer supplies geospatial information.

7.1 Web Map Service (WMS)

This section describes the Web Map Service (WMS).

7.1.1 WMS settings

This page details the configuration options for WMS in the web administration interface.

Service Metadata

See the section on *Service Metadata*.

Root Layer Information

In this section is possible to define a title and an abstract for the root layer in the WMS capabilities. When these are left empty the WMS service title and abstract are used.

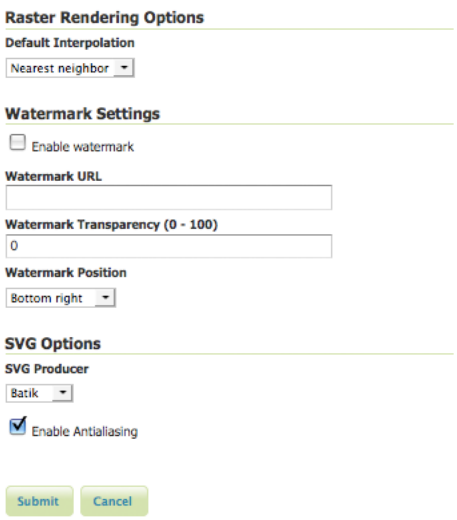


Fig. 7.1: WMS configuration options

Raster Rendering Options

The Web Map Service Interface Standard (WMS) provides a simple way to request and serve geo-registered map images. During pan and zoom operations, WMS requests generate map images through a variety of raster rendering processes. Such image manipulation is generally called resampling, interpolation, or down-sampling. GeoServer supports three resampling methods that determine how cell values of a raster are outputted. These sampling methods—Nearest Neighbor, Bilinear Interpolation and Bicubic—are available on the Default Interpolation menu.

Nearest Neighbor—Uses the center of nearest input cell to determine the value of the output cell. Original values are retained and no new averages are created. Because image values stay exactly the same, rendering is fast but possibly pixelated from sharp edge detail. Nearest neighbor interpolation is recommended for categorical data such as land use classification.

Bilinear—Determines the value of the output cell based by sampling the value of the four nearest cells by linear weighting. The closer an input cell, the higher its influence of on the output cell value. Since output values may differ from nearest input, bilinear interpolation is recommended for continuous data like elevation and raw slope values. Bilinear interpolation takes about five times as long as nearest neighbor interpolation.

Bicubic—Looks at the sixteen nearest cells and fits a smooth curve through the points to find the output value. Bicubic interpolation may both change the input value as well as place the output value outside of the range of input values. Bicubic interpolation is recommended for smoothing continuous data, but this incurs a processing performance overhead.

Watermark Settings

Watermarking is the process of embedding an image into a map. Watermarks are usually used for branding, copyright, and security measures. Watermarks are configured in the WMS watermarks setting section.

Enable Watermark—Turns on watermarking. When selected, all maps will render with the same watermark. It is not currently possible to specify watermarking on a per-layer or per-feature basis.

Watermark URL—Location of the graphic for the watermark. The graphic can be referenced as an absolute path (e.g., `C:\GeoServer\watermark.png`), a relative one inside GeoServer's data directory (e.g., `watermark.png`), or a URL (e.g., `http://www.example.com/images/watermark.png`).

Each of these methods have their own advantages and disadvantages. When using an absolute or relative link, GeoServer keeps a cached copy of the graphic in memory, and won't continually link to the original file. This means that if the original file is subsequently deleted, GeoServer won't register it missing until the watermark settings are edited. Using a URL might seem more convenient, but it is more I/O intensive. GeoServer will load the watermark image for every WMS request. Also, should the URL cease to be valid, the layer will not properly display.

Watermark Transparency–Determines the opacity level of the watermark. Numbers range between 0 (opaque) and 100 (fully invisible).

Watermark Position—Specifies the position of the watermark relative to the WMS request. The nine options indicate which side and corner to place the graphic (top-left, top-center, top-right, etc). The default watermark position is bottom-right. Note that the watermark will always be displayed flush with the boundary. If extra space is required, the graphic itself needs to change.

Because each WMS request renders the watermark, a single tiled map positions *one* watermark relative to the view window while a tiled map positions the watermark for each tile. The only layer specific aspect of watermarking occurs because a single tile map is one WMS request, whereas a tiled map contains many WMS requests. (The latter watermark display resembles Google Maps faint copyright notice in their Satellite imagery.) The following three examples demonstrate watermark position, transparency and tiling display, respectively.

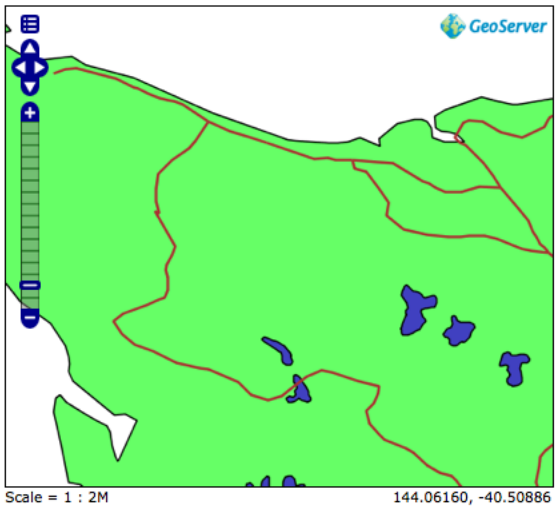


Fig. 7.2: Single tile watermark (aligned top-right, transparency=0)

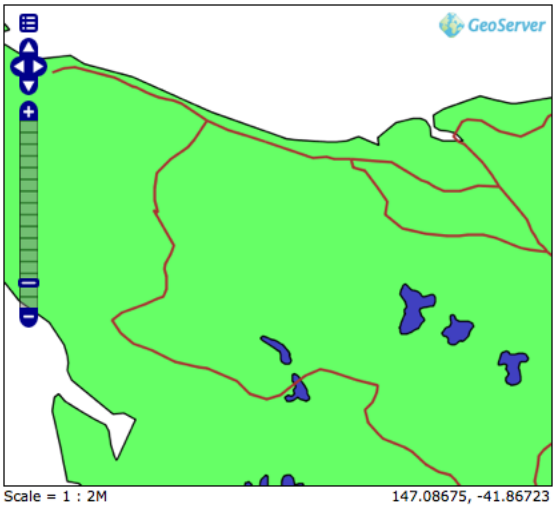


Fig. 7.3: Single tile watermark (aligned top-right, transparency=90)

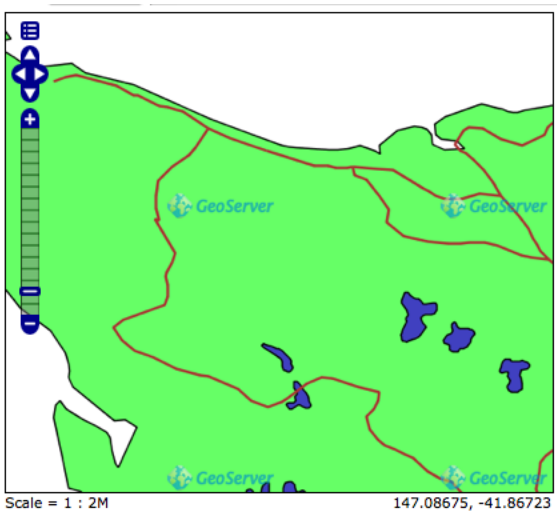


Fig. 7.4: Tiled watermark (aligned top-right, transparency=90)

SVG Options

The GeoServer WMS supports SVG (Scalable Vector Graphics) as an output format. GeoServer currently supports two SVG renderers, available from the SVG producer menu.

1. *Simple*—Simple SVG renderer. It has limited support for SLD styling, but is very fast.
2. *Batik*—Batik renderer (as it uses the Batik SVG Framework). It has full support for SLD styling, but is slower.

Enable Anti-aliasing Anti-aliasing is a technique for making edges appear smoother by filling in the edges of an object with pixels that are between the object's color and the background color. Anti-aliasing creates the illusion of smoother lines and smoother selections. Turning on anti-aliasing will generally make maps look nicer, but will increase the size of the images, and will take longer to return. If you are overlaying the anti-aliased map on top of others, beware of using transparencies as the anti-aliasing process mixes with the colors behind and can create a "halo" effect.

Advanced projection handling and map wrapping

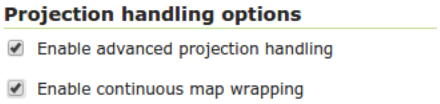
Advanced projection handling is a set of extra "smarts" applied while rendering that help getting a good looking map despite the data touching or crossing "difficult areas" in selected map projection. This includes, among others:

- Cutting the geometries so that they fit within the area of mathematical stability of the projection math, e.g., it will cut any bit at more than 45 degrees west and east from the central meridian of a transverse Mercator projection, or beyond 85 degrees north or south in a Mercator projection
- Make sure both "ends" of the world get queried for data when a map in polar stereographic is hitting an area that includes the dateline

Along with advanced projection handling there is the possibility of creating a continuous map across the dateline, wrapping the data on the other side of the longitude range, to get a continuous map. This is called continuous map wrapping, and it's enabled in Mercator and Equirectangular (plate carrée) projections.

Both functionalities are rather useful, and enabled by default, but the tendency to generate multiple ored bounding boxes (to query both sides of the dateline) can cause extreme slowness in certain databases

(e.g. Oracle), and some users might simply not like the wrapping output, thus, it's possible to disable both functions in the WMS UI:

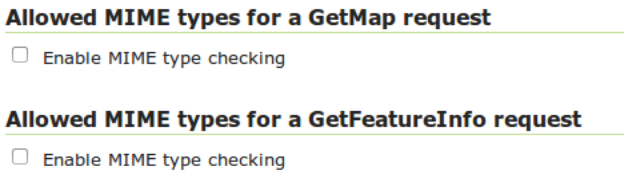


Continuous map wrapping is disabled if advanced projection handling is disabled.

Advanced projection handling can also be disabled using the `advancedProjectionHandling` *Format Option*. Similarly, continuous map wrapping can also be disabled using the `mapWrapping` *Format Option*.

Restricting MIME types for GetMap and GetFeatureInfo requests

GeoServer supports restricting formats for WMS GetMap and WMS GetFeatureInfo requests. The default is to allow all MIME types for both kinds of request.



The following figure shows an example for MIME type restriction. The MIME types **image/png** and **text/html;subtype=openlayers** are allowed for GetMap requests, the MIME types **text/html** and **text/plain** are allowed for GetFeatureInfo requests. A GetMap/GetFeatureInfo request with a MIME type not allowed will result in a service exception reporting the error.

Note: Activating MIME type restriction and not allowing at least one MIME type disables the particular request.

Disabling usage of dynamic styling in GetMap and GetFeatureInfo requests

Dynamic styles can be applied to layers in GetMap and GetFeatureInfo requests using the SLD or SLD_BODY parameters for GET requests.

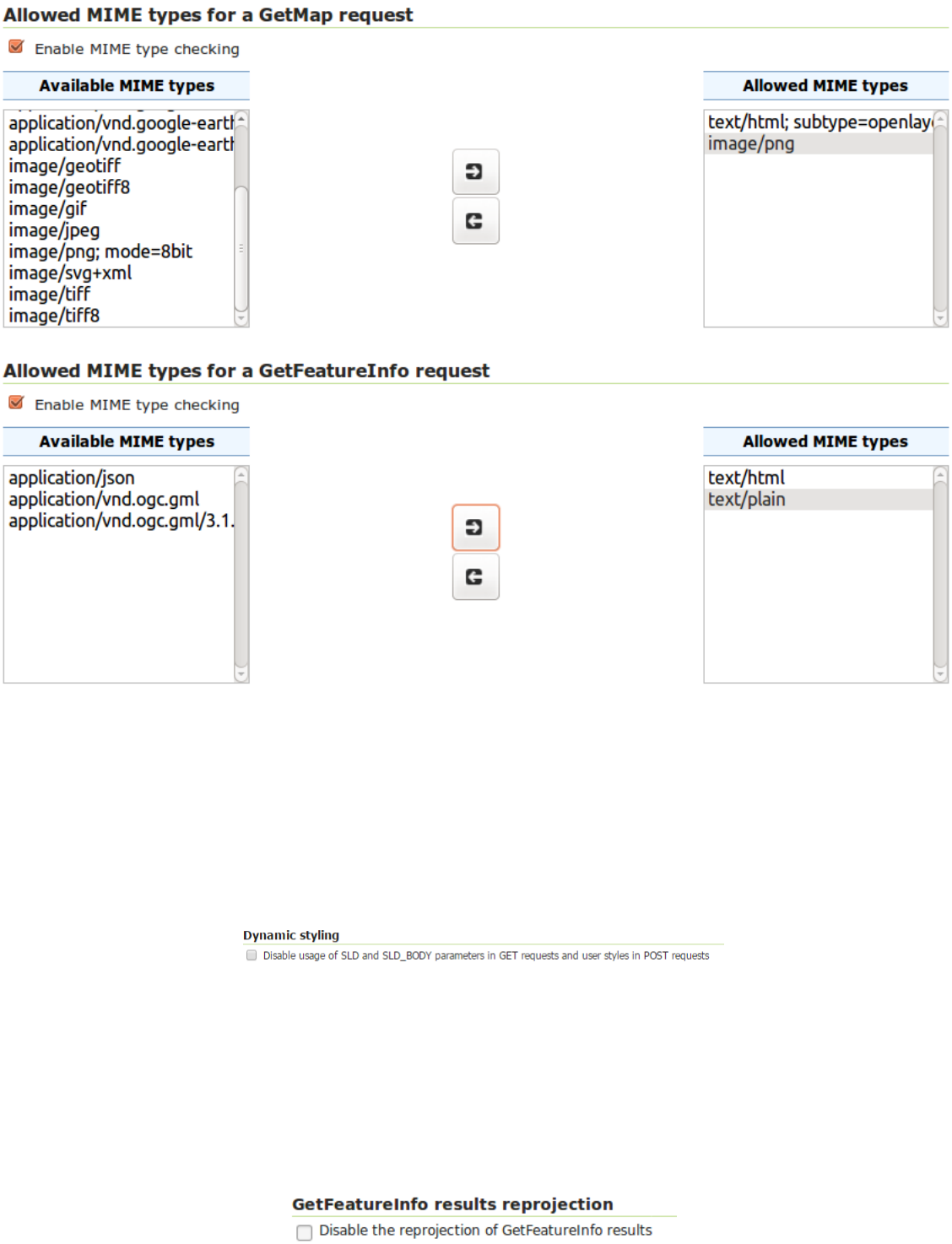
In addition, GetMap POST requests can contain inline style definition for layers.

The usage of dynamic styling can be restricted on a global or per virtual service basis using the **Dynamic styling** section.

When the flag is checked, a GetMap/GetFeatureInfo request with a dynamic style will result in a service exception reporting the error.

Disabling GetFeatureInfo requests results reprojection

By default GetFeatureInfo results are reproject to the map coordinate reference system. This behavior can be deactivated on a global or per virtual service basis in the **GetFeatureInfo results reprojection** section.


Allowed MIME types for a GetMap request
Enable MIME type checking
Available MIME types
application/vnd.google-earth
application/vnd.google-earth
image/geotiff
image/geotiff8
image/gif
image/jpeg
image/png; mode=8bit
image/svg+xml
image/tiff
image/tiff8
Allowed MIME types
text/html; subtype=openlay
image/png
Allowed MIME types for a GetFeatureInfo request
Enable MIME type checking
Available MIME types
application/json
application/vnd.ogc.gml
application/vnd.ogc.gml/3.1.
Allowed MIME types
text/html
text/plain
Dynamic styling
Disable usage of SLD and SLD_BODY parameters in GET requests and user styles in POST requests
GetFeatureInfo results reprojection
Disable the reprojection of GetFeatureInfo results

When the flag is checked, GetFeatureInfo requests results will not be reprojected and will instead used the layer coordinate reference system.

7.1.2 WMS basics

GeoServer provides support for Open Geospatial Consortium (OGC) **Web Map Service (WMS)** versions 1.1.1 and 1.3.0. This is the most widely used standard for generating maps on the web, and it is the primary interface to request map products from GeoServer. Using WMS makes it possible for clients to overlay maps from several different sources in a seamless way.

GeoServer's WMS implementation fully supports the standard, and is certified compliant against the OGC's test suite. It includes a wide variety of rendering and labeling options, and is one of the fastest WMS Servers for both raster and vector data.

GeoServer WMS supports reprojection to any **coordinate reference system** in the EPSG database. It is possible to add additional coordinate systems if the Well Known Text definition is known. See *Coordinate Reference System Handling* for details.

GeoServer fully supports the **Styled Layer Descriptor (SLD)** standard, and uses SLD files as its native styling language. For more information on how to style data in GeoServer see the section *Styling*

Differences between WMS versions

The major differences between versions 1.1.1 and 1.3.0 are:

- In 1.1.1 geographic coordinate systems specified with the `EPSG` namespace are defined to have an axis ordering of longitude/latitude. In 1.3.0 the ordering is latitude/longitude. See *Axis Ordering* below for more details.
- In the GetMap operation the `srs` parameter is called `crs` in 1.3.0. GeoServer supports both keys regardless of version.
- In the GetFeatureInfo operation the `x` and `y` parameters are called `i` and `j` in 1.3.0. GeoServer supports both keys regardless of version, except when in CITE-compliance mode.

Axis Ordering

The WMS 1.3.0 specification mandates that the axis ordering for geographic coordinate systems defined in the EPSG database be *latitude/longitude*, or *y/x*. This is contrary to the fact that most spatial data is usually in *longitude/latitude*, or *x/y*. This requires that the coordinate order in the `BBOX` parameter be reversed for `SRS` values which are geographic coordinate systems.

For example, consider the WMS 1.1 request using the WGS84 SRS (EPSG:4326):

```
geoserver/wms?VERSION=1.1.1&REQUEST=GetMap&SRS=epsg:4326&BBOX=-180,-90,180,90&...
```

The equivalent WMS 1.3.0 request is:

```
geoserver/wms?VERSION=1.3.0&REQUEST=GetMap&CRS=epsg:4326&BBOX=-90,-180,90,180&...
```

Note that the coordinates specified in the `BBOX` parameter are reversed.

7.1.3 WMS reference

Introduction

The OGC Web Map Service (WMS) specification defines an HTTP interface for requesting georeferenced map images from a server. GeoServer supports WMS 1.1.1, the most widely used version of WMS, as well as WMS 1.3.0.

The relevant OGC WMS specifications are:

- OGC Web Map Service Implementation Specification, Version 1.1.1
- OGC Web Map Service Implementation Specification, Version 1.3.0

GeoServer also supports some extensions to the WMS specification made by the Styled Layer Descriptor (SLD) standard to control the styling of the map output. These are defined in:

- OpenGIS Styled Layer Descriptor Profile of the Web Map Service Implementation Specification, Version 1.1.0

Benefits of WMS

WMS provides a standard interface for requesting a geospatial map image. The benefit of this is that WMS clients can request images from multiple WMS servers, and then combine them into a single view for the user. The standard guarantees that these images can all be overlaid on one another as they actually would be in reality. Numerous servers and clients support WMS.

Operations

WMS requests can perform the following operations:

Operation	Description
`Exceptions`	If an exception occur
`GetCapabilities`	Retrieves metadata about the service, including supported operations and parameters, and a list of the available layers
`GetMap`	Retrieves a map image for a specified area and content
`GetFeatureInfo` (optional)	Retrieves the underlying data, including geometry and attribute values, for a pixel location on a map
`DescribeLayer` (optional)	Indicates the WFS or WCS to retrieve additional information about the layer.
`GetLegendGraphic` (optional)	Retrieves a generated legend for a map

Exceptions

Formats in which WMS can report exceptions. The supported values for exceptions are:

Format	Syntax	Notes
XML	`EXCEPTIONS=application/vnd.ogc.se_xml`	Xml output. (The default format)
INIMAGE	`EXCEPTIONS=application/vnd.ogc.se_inimage`	Generates an image
BLANK	`EXCEPTIONS=application/vnd.ogc.se_blank`	Generates a blank image
PARTIALMAP	`EXCEPTIONS=application/vnd.gs.wms_partial`	This is a GeoServer vendor parameter and only applicable for getMap requests. Returns everything that was rendered at the time the rendering process threw an exception. Can be used with the *WMS Configuration Limits* to return a partial image even if the request is terminated for exceeding one of these limits. It also works with the `timeout` *vendor parameter*.
JSON	`EXCEPTIONS=application/json`	Simple Json representation.
JSONP	`EXCEPTIONS=text/javascript`	Return a JsonP in the form: paddingOutput(...jsonp...). See *WMS vendor parameters* to change the callback name. Note that this format is disabled by default (See *Global variables affecting WMS*).

GetCapabilities

The **GetCapabilities** operation requests metadata about the operations, services, and data ("capabilities") that are offered by a WMS server.

The parameters for the GetCapabilities operation are:

Parameter	Required?	Description
`service`	Yes	Service name. Value is `WMS`.
`version`	Yes	Service version. Value is one of `1.0.0`, `1.1.0`, `1.1.1`, `1.3.0`.
`request`	Yes	Operation name. Value is `GetCapabilities`.

GeoServer provides the following vendor-specific parameters for the GetCapabilities operation. They are fully documented in the *WMS vendor parameters* section.

Parameter	Required?	Description
`namespace`	No	limits response to layers in a given namespace
`format`	No	request the capabilities document in a certain format

A example GetCapabilities request is:

```
http://localhost:8080/geoserver/wms?
service=wms&
version=1.1.1&
request=GetCapabilities
```

There are three parameters being passed to the WMS server, `service=wms`, `version=1.1.1`, and `request=GetCapabilities`. The `service` parameter tells the WMS server that a WMS request is forthcoming. The `version` parameter refers to which version of WMS is being requested. The `request` pa-

rameter specifies the GetCapabilities operation. The WMS standard requires that requests always includes these three parameters. GeoServer relaxes these requirements (by setting the default version if omitted), but for standard-compliance they should always be specified.

The response is a Capabilities XML document that is a detailed description of the WMS service. It contains three main sections:

Service	Contains service metadata such as the service name, keywords, and contact information for the organization operating the server.
Request	Describes the operations the WMS service provides and the parameters and output formats for each operation. If desired GeoServer can be configured to disable support for certain WMS operations.
Layer	Lists the available coordinate systems and layers. In GeoServer layers are named in the form "namespace:layer". Each layer provides service metadata such as title, abstract and keywords.

GetMap

The **GetMap** operation requests that the server generate a map. The core parameters specify one or more layers and styles to appear on the map, a bounding box for the map extent, a target spatial reference system, and a width, height, and format for the output. The information needed to specify values for parameters such as `layers`, `styles` and `srs` can be obtained from the Capabilities document.

The response is a map image, or other map output artifact, depending on the format requested. GeoServer provides a wide variety of output formats, described in *WMS output formats*.

The standard parameters for the GetMap operation are:

Parameter	Required?	Description
service	Yes	Service name. Value is WMS.
version	Yes	Service version. Value is one of 1.0.0, 1.1.0, 1.1.1, 1.3.0.
request	Yes	Operation name. Value is GetMap.
layers	Yes	Layers to display on map. Value is a comma-separated list of layer names.
styles	Yes	Styles in which layers are to be rendered. Value is a comma-separated list of style names, or empty if default styling is required. Style names may be empty in the list, to use default layer styling.
srs *or* crs	Yes	Spatial Reference System for map output. Value is in form EPSG:nnn. crs is the parameter key used in WMS 1.3.0.
bbox	Yes	Bounding box for map extent. Value is minx,miny,maxx,maxy in units of the SRS.
width	Yes	Width of map output, in pixels.
height	Yes	Height of map output, in pixels.
format	Yes	Format for the map output. See *WMS output formats* for supported values.
transparent	No	Whether the map background should be transparent. Values are true or false. Default is false
bgcolor	No	Background color for the map image. Value is in the form RRGGBB. Default is FFFFFF (white).
exceptions	No	Format in which to report exceptions. Default value is application/vnd.ogc.se_xml.
time	No	Time value or range for map data. See *Time Support in GeoServer WMS* for more information.
sld	No	A URL referencing a *StyledLayerDescriptor* XML file which controls or enhances map layers and styling
sld_body	No	A URL-encoded *StyledLayerDescriptor* XML document which controls or enhances map layers and styling

GeoServer provides a number of useful vendor-specific parameters for the GetMap operation. These are documented in the *WMS vendor parameters* section.

Example WMS request for topp:states layer to be output as a PNG map image in SRS EPGS:4326 and using default styling is:

```
http://localhost:8080/geoserver/wms?
request=GetMap
&service=WMS
&version=1.1.1
&layers=topp%3Astates
&styles=population
&srs=EPSG%3A4326
&bbox=-145.15104058007,21.731919794922,-57.154894212888,58.961058642578&
&width=780
&height=330
&format=image%2Fpng
```

The standard specifies many of the parameters as being mandatory, GeoServer provides the *WMS Reflector* to allow many of them to be optionally specified.

Experimenting with this feature is a good way to get to know the GetMap parameters.

Example WMS request using a GetMap XML document is:

```
<?xml version="1.0" encoding="UTF-8"?>
<ogc:GetMap xmlns:ogc="http://www.opengis.net/ows"
            xmlns:gml="http://www.opengis.net/gml"
   version="1.1.1" service="WMS">
   <StyledLayerDescriptor version="1.0.0">
      <NamedLayer>
        <Name>topp:states</Name>
        <NamedStyle><Name>population</Name></NamedStyle>
      </NamedLayer>
   </StyledLayerDescriptor>
   <BoundingBox srsName="http://www.opengis.net/gml/srs/epsg.xml#4326">
      <gml:coord><gml:X>-130</gml:X><gml:Y>24</gml:Y></gml:coord>
      <gml:coord><gml:X>-55</gml:X><gml:Y>50</gml:Y></gml:coord>
   </BoundingBox>
   <Output>
      <Format>image/png</Format>
      <Size><Width>550</Width><Height>250</Height></Size>
   </Output>
</ogc:GetMap>
```

Time

As of GeoServer 2.2.0, GeoServer supports a TIME attribute for WMS GetMap requests as described in version 1.3.0 of the WMS specification. This parameter allows filtering a dataset by temporal slices as well as spatial tiles for rendering. See *Time Support in GeoServer WMS* for information on its use.

GetFeatureInfo

The **GetFeatureInfo** operation requests the spatial and attribute data for the features at a given location on a map. It is similar to the WFS *GetFeature* operation, but less flexible in both input and output. Since GeoServer provides a WFS service we recommend using it instead of `GetFeatureInfo` whenever possible.

The one advantage of `GetFeatureInfo` is that the request uses an (x,y) pixel value from a returned WMS image. This is easier to use for a naive client that is not able to perform true geographic referencing.

The standard parameters for the GetFeatureInfo operation are:

Parameter	Required?	Description
service	Yes	Service name. Value is WMS.
version	Yes	Service version. Value is one of 1.0.0, 1.1.0, 1.1.1, 1.3.0.
request	Yes	Operation name. Value is GetFeatureInfo.
layers	Yes	See *GetMap*
styles	Yes	See *GetMap*
srs *or* crs	Yes	See *GetMap*
bbox	Yes	See *GetMap*
width	Yes	See *GetMap*
height	Yes	See *GetMap*
query_layers	Yes	Comma-separated list of one or more layers to query.
info_format	No	Format for the feature information response. See below for values.
feature_count	No	Maximum number of features to return. Default is 1.
x or i	Yes	X ordinate of query point on map, in pixels. 0 is left side. i is the parameter key used in WMS 1.3.0.
y or j	Yes	Y ordinate of query point on map, in pixels. 0 is the top. j is the parameter key used in WMS 1.3.0.
exceptions	No	Format in which to report exceptions. The default value is application/vnd.ogc.se_xml.

Note: If you are sending a GetFeatureInfo request against a layergroup, all the layers in that layergroup must be set as "Queryable" to get a result (See *WMS Settings on Layers page*)

GeoServer supports a number of output formats for the GetFeatureInfo response. Server-styled HTML is the most commonly-used format. For maximum control and customisation the client should use GML3 and style the raw data itself. The supported formats are:

Format	Syntax	Notes
TEXT	info_format=text/plain	Simple text output. (The default format)
GML 2	info_format=application/vnd.ogc.gml	Works only for Simple Features (see *Complex Features*)
GML 3	info_format=application/vnd.ogc.gml/3.1.1	Works for both Simple and Complex Features (see *Complex Features*)
HTML	info_format=text/html	Uses HTML templates that are defined on the server. See *GetFeatureInfo Templates* for information on how to template HTML output.
JSON	info_format=application/json	Simple Json representation.
JSONP	info_format=text/javascript	Returns a JsonP in the form: parseResponse(...json...). See *WMS vendor parameters* to change the callback name. Note that this format is disabled by default (See *Global variables affecting WMS*).

GeoServer provides the following vendor-specific parameters for the GetFeatureInfo operation. They are fully documented in the *WMS vendor parameters* section.

Parameter	Required?	Description
buffer	No	width of search radius around query point.
cql_filter	No	Filter for returned data, in ECQL format
filter	No	Filter for returned data, in OGC Filter format
propertyName	No	Feature properties to be returned

An example request for feature information from the `topp:states` layer in HTML format is:

```
http://localhost:8080/geoserver/wms?
request=GetFeatureInfo
&service=WMS
&version=1.1.1
&layers=topp%3Astates
&styles=
&srs=EPSG%3A4326
&format=image%2Fpng
&bbox=-145.151041%2C21.73192%2C-57.154894%2C58.961059
&width=780
&height=330
&query_layers=topp%3Astates
&info_format=text%2Fhtml
&feature_count=50
&x=353
&y=145
&exceptions=application%2Fvnd.ogc.se_xml
```

An example request for feature information in GeoJSON format is:

```
http://localhost:8080/geoserver/wms?
&INFO_FORMAT=application/json
&REQUEST=GetFeatureInfo
&EXCEPTIONS=application/vnd.ogc.se_xml
&SERVICE=WMS
&VERSION=1.1.1
&WIDTH=970&HEIGHT=485&X=486&Y=165&BBOX=-180,-90,180,90
&LAYERS=COUNTRYPROFILES:grp_administrative_map
&QUERY_LAYERS=COUNTRYPROFILES:grp_administrative_map
&TYPENAME=COUNTRYPROFILES:grp_administrative_map
```

The result will be:

```
{
"type":"FeatureCollection",
"features":[
   {
      "type":"Feature",
      "id":"dt_gaul_geom.fid-138e3070879",
      "geometry":{
         "type":"MultiPolygon",
         "coordinates":[
            [
               [
                  [
                     XXXXXXXXXX,
                     XXXXXXXXXX
                  ],
                  ...
                  [
                     XXXXXXXXXX,
                     XXXXXXXXXX
                  ]
               ]
            ]
         ]
      },
```

```
      "geometry_name":"at_geom",
      "properties":{
         "bk_gaul":X,
         "at_admlevel":0,
         "at_iso3":"XXX",
         "ia_name":"XXXX",
         "at_gaul_l0":X,
         "bbox":[
            XXXX,
            XXXX,
            XXXX,
            XXXX
         ]
      }
   }
],
"crs":{
   "type":"EPSG",
   "properties":{
      "code":"4326"
   }
},
"bbox":[
   XXXX,
   XXXX,
   XXXX,
   XXXX
]
}
```

DescribeLayer

The **DescribeLayer** operation is used primarily by clients that understand SLD-based WMS. In order to make an SLD one needs to know the structure of the data. WMS and WFS both have operations to do this, so the **DescribeLayer** operation just routes the client to the appropriate service.

The standard parameters for the DescribeLayer operation are:

Parameter	Required?	Description
`service`	Yes	Service name. Value is `WMS`.
`version`	Yes	Service version. Value is `1.1.1`.
`request`	Yes	Operation name. Value is `DescribeLayer`.
`layers`	Yes	See *GetMap*
`exceptions`	No	Format in which to report exceptions. The default value is `application/vnd.ogc.se_xml`.

GeoServer supports a number of output formats for the `DescribeLayer` response. Server-styled HTML is the most commonly-used format. The supported formats are:

Format	Syntax	Notes
TEXT	`output_format=text/xml`	Same as default.
GML 2	`output_format=application/vnd.ogc.wms_xml`	The default format.
JSON	`output_format=application/json`	Simple Json representation.
JSONP	`output_format=text/javascript`	Return a JsonP in the form: paddingOutput(...jsonp...). See *WMS vendor parameters* to change the callback name. Note that this format is disabled by default (See *Global variables affecting WMS*).

An example request in XML (default) format on a layer is: :

http://localhost:8080/geoserver/topp/wms?service=WMS &version=1.1.1 &request=DescribeLayer &layers=topp:coverage

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE WMS_DescribeLayerResponse SYSTEM "http://localhost:8080/geoserver/schemas/
↪wms/1.1.1/WMS_DescribeLayerResponse.dtd">
<WMS_DescribeLayerResponse version="1.1.1">
   <LayerDescription name="topp:coverage" owsURL="http://localhost:8080/geoserver/
↪topp/wcs?" owsType="WCS">
      <Query typeName="topp:coverage"/>
   </LayerDescription>
</WMS_DescribeLayerResponse>
```

An example request for feature description in JSON format on a layer group is:

```
http://localhost:8080/geoserver/wms?service=WMS
&version=1.1.1
&request=DescribeLayer
&layers=sf:roads,topp:tasmania_roads,nurc:mosaic
&outputFormat=application/json
```

The result will be:

```
{
  version: "1.1.1",
  layerDescriptions: [
    {
        layerName: "sf:roads",
        owsURL: "http://localhost:8080/geoserver/wfs/WfsDispatcher?",
        owsType: "WFS",
        typeName: "sf:roads"
    },
    {
        layerName: "topp:tasmania_roads",
        owsURL: "http://localhost:8080/geoserver/wfs/WfsDispatcher?",
        owsType: "WFS",
        typeName: "topp:tasmania_roads"
    },
    {
        layerName: "nurc:mosaic",
        owsURL: "http://localhost:8080/geoserver/wcs?",
        owsType: "WCS",
        typeName: "nurc:mosaic"
```

```
    }
  ]
```

GetLegendGraphic

The **GetLegendGraphic** operation provides a mechanism for generating legend graphics as images, beyond the LegendURL reference of WMS Capabilities. It generates a legend based on the style defined on the server, or alternatively based on a user-supplied SLD. For more information on this operation and the various options that GeoServer supports see *GetLegendGraphic*.

7.1.4 Time Support in GeoServer WMS

GeoServer supports a `TIME` attribute in GetMap requests for layers that are properly configured with a time dimension. This is used to specify a temporal subset for rendering.

For example, you might have a single dataset with weather observations collected over time and choose to plot a single day's worth of observations.

The attribute to be used in `TIME` requests can be set up through the GeoServer web interface by navigating to *Layers -> [specific layer] -> Dimensions tab*.

Note: Read more about how to *use the web interface to configure an attribute for TIME requests*.

Specifying a time

The format used for specifying a time in the WMS TIME parameter is based on ISO-8601. Times may be specified up to a precision of 1 millisecond; GeoServer does not represent time queries with more precision than this.

The parameter is:

```
TIME=<timestring>
```

Times follow the general format:

```
yyyy-MM-ddThh:mm:ss.SSSZ
```

where:

- `yyyy`: 4-digit year
- `MM`: 2-digit month
- `dd`: 2-digit day
- `hh`: 2-digit hour
- `mm`: 2-digit minute
- `ss`: 2-digit second
- `SSS`: 3-digit millisecond

The day and intraday values are separated with a capital `T`, and the entire thing is suffixed with a `Z`, indicating UTC for the time zone. (The WMS specification does not provide for other time zones.)

GeoServer will apply the `TIME` value to all temporally enabled layers in the `LAYERS` parameter of the GetMap request. Layers without a temporal component will be served normally, allowing clients to include reference information like political boundaries along with temporal data.

Description	Time specification
December 12, 2001 at 6:00 PM	`2001-12-12T18:00:00.0Z`
May 5, 1993 at 11:34 PM	`1993-05-05T11:34:00.0Z`

Specifying an absolute interval

A client may request information over a continuous interval instead of a single instant by specifying a start and end time, separated by a `/` character.

In this scenario the start and end are *inclusive*; that is, samples from exactly the endpoints of the specified range will be included in the rendered tile.

Description	Time specification
The month of September 2002	`2002-09-01T00:00:00.0Z/2002-09-30T23:59:59.999Z`
The entire day of December 25, 2010	`2010-12-25T00:00:00.0Z/2010-12-25T23:59:59.999Z`

Specifying a relative interval

A client may request information over a relative time interval instead of a set time range by specifying a start or end time with an associated duration, separated by a `/` character.

One end of the interval must be a time value, but the other may be a duration value as defined by the ISO 8601 standard. The special keyword `PRESENT` may be used to specify a time relative to the present server time.

Description	Time specification
The month of September 2002	`2002-09-01T00:00:00.0Z/P1M`
The entire day of December 25, 2010	`2010-12-25T00:00:00.0Z/P1D`
The entire day preceding December 25, 2010	`P1D/2010-12-25T00:00:00.0Z`
36 hours preceding the current time	`PT36H/PRESENT`

Note: The final example could be paired with the KML service to provide a *Google Earth* network link which is always updated with the last 36 hours of data.

Reduced accuracy times

The WMS specification also allows time specifications to be truncated by omitting some of the time string. In this case, GeoServer treats the time as a range whose length is equal to the *most precise unit specified* in the time string.

For example, if the time specification omits all fields except year, it identifies a range one year long starting at the beginning of that year.

Note: GeoServer implements this by adding the appropriate unit, then subtracting 1 millisecond. This avoids surprising results when using an interval that aligns with the actual sampling frequency of the data - for example, if yearly data is natively stored with dates like 2001-01-01T00:00:00.0Z, 2002-01-01T00:00:00Z, etc. then a request for 2001 would include the samples for both 2001 and 2002, which wouldn't be desired.

Description	Reduced Accuracy Time	Equivalent Range
The month of September 2002	`2002-09`	`2002-09-01T00:00:00.0Z/ 2002-09-30T23:59:59.999Z`
The day of December 25, 2010	`2010-12-25`	`2010-12-25T00:00:00.0Z/ 2010-12-25T23:59:59.999Z`

Reduced accuracy times with ranges

Reduced accuracy times are also allowed when specifying ranges. In this case, GeoServer effectively expands the start and end times as described above, and then includes any samples from after the beginning of the start interval and before the end of the end interval.

Note: Again, the ranges are inclusive.

Description	Reduced Accuracy Time	Equivalent Range
The months of September through December 2002	`2002-09/2002-12`	`2002-09-01T00:00:00.0Z/ 2002-12-31T23:59:59.999Z`
12PM through 6PM, December 25, 2010	`2010-12-25T12/ 2010-12-25T18`	`2010-12-25T12:00:00.0Z/ 2010-12-25T18:59:59.999Z`

Note: In the last example, note that the result may not be intuitive, as it includes all times from 6PM to 6:59PM.

Specifying a list of times

GooServer can also accept a list of discrete time values. This is useful for some applications such as animations, where one time is equal to one frame.

The elements of a list are separated by commas.

Note: GeoServer currently does not support lists of ranges, so all list queries effectively have a resolution of 1 millisecond. If you use reduced accuracy notation when specifying a range, each range will be automatically converted to the instant at the beginning of the range.

If the list is evenly spaced (for example, daily or hourly samples) then the list may be specified as a range, using a start time, end time, and period separated by slashes.

Description	List notation	Equivalent range notation
Noon every day for August 12-14, 2012	`TIME=2012-08-12T12:00:00.0Z,2012-08-13T12:00:00.0Z,2012-08-14T12:00:00.0Z`	`TIME=2012-08-12T12:00:00.0Z/2012-08-18:T12:00:00.0Z/P1D`
Midnight on the first of September, October, and November 1999	`TIME=1999-09-01T00:00:00.0Z,1999-10-01T00:00:00.0Z,1999-11-01T00:00:00.0Z`	`TIME=1999-09-01T00:00:00.0Z/1999-11-01T00:00:00.0Z/P1M`

Specifying a periodicity

The periodicity is also specified in ISO-8601 format: a capital P followed by one or more interval lengths, each consisting of a number and a letter identifying a time unit:

Unit	Abbreviation
Years	`Y`
Months	`M`
Days	`D`
Hours	`H`
Minutes	`M`
Seconds	`S`

The Year/Month/Day group of values must be separated from the Hours/Minutes/Seconds group by a `T` character. The T itself may be omitted if hours, minutes, and seconds are all omitted. Additionally, fields which contain a 0 may be omitted entirely.

Fractional values are permitted, but only for the most specific value that is included.

Note: The period must divide evenly into the interval defined by the start/end times. So if the start/end times denote 12 hours, a period of 1 hour would be allowed, but a period of 5 hours would not.

For example, the multiple representations listed below are all equivalent.

- One hour:

```
P0Y0M0DT1H0M0S

PT1H0M0S

PT1H
```

- 90 minutes:

```
P0Y0M0DT1H30M0S

PT1H30M

P90M
```

- 18 months:

```
P1Y6M0DT0H0M0S

P1Y6M0D

P0Y18M0DT0H0M0S

P18M
```

Note: `P1.25Y3M` would not be acceptable, because fractional values are only permitted in the most specific value given, which in this case would be months.

7.1.5 WMS output formats

WMS returns images in a number of possible formats. This page shows a list of the output formats. The syntax for setting an output format is:

```
format=<format>
```

where `<format>` is any of the options below.

Note: The list of output formats supported by a GeoServer instance can be found by a WMS *GetCapabilities* request.

Format	Syntax	Notes
PNG	`format=image/png`	Default
PNG8	`format=image/png8`	Same as PNG, but computes an optimal 256 color (8 bit) palette, so the image size is usually smaller
JPEG	`format=image/jpeg`	
JPEG-PNG	`format=image/vnd.jpeg-png`	A custom format that will decide dynamically, based on the image contents, if it's best to use a JPEG or PNG compression. The images are returned in JPEG format if fully opaque and not paletted. In order to use this format in a meaningful way the GetMap must include a "&transparent=TRUE" parameter, as without it GeoServer generates opaque images with the default/requested background color, making this format always return JPEG images (or always PNG, if they are paletted). When using the layer preview to test this format, remember to add "&transparent=TRUE" to the preview URL, as normally the preview generates non transparent images.
JPEG-PNG8	`format=image/vnd.jpeg-png8`	Same as JPEG-PNG, but generating a paletted output if the PNG format is chosen
GIF	`format=image/gif`	
TIFF	`format=image/tiff`	
TIFF8	`format=image/tiff8`	Same as TIFF, but computes an optimal 256 color (8 bit) palette, so the image size is usually smaller
GeoTIFF	`format=image/geotiff`	Same as TIFF, but includes extra GeoTIFF metadata
GeoTIFF8	`format=image/geotiff8`	Same as TIFF, but includes extra GeoTIFF metadata and computes an optimal 256 color (8 bit) palette, so the image size is usually smaller
SVG	`format=image/svg`	
PDF	`format=application/pdf`	
GeoRSS	`format=rss`	
KML	`format=kml`	
KMZ	`format=kmz`	
OpenLayers	`format=application/openlayers`	Generates an OpenLayers HTML application.
UTFGrid	`format=application/json;type=utfgrid`	Generates an UTFGrid 1.3 JSON response. Requires vector output, either from a vector layer, or from a raster layer turned into vectors by a rendering transformation.

7.1.6 WMS vendor parameters

WMS vendor parameters are non-standard request parameters that are defined by an implementation to provide enhanced capabilities. GeoServer supports a variety of vendor-specific WMS parameters.

angle

The `angle` parameter rotates the output map clockwise around its center. The syntax is:

```
angle=<x>
```

where `<x>` is the number of degrees to rotate by.

Map rotation is supported in all raster formats, PDF, and SVG when using the Batik producer (which is the default).

buffer

The `buffer` parameter specifies the number of additional border pixels that are used in the `GetMap` and `GetFeatureInfo` operations. The syntax is:

```
buffer=<bufferwidth>
```

where `<bufferwidth>` is the width of the buffer in pixels.

In the *GetMap* operation, buffering includes features that lie outside the request bounding box, but whose styling is thick enough to be visible inside the map area.

In the *GetFeatureInfo* operation, buffering creates a "search radius" around the location of the request. Feature info is returned for features intersecting the search area. This is useful when working with an OpenLayers map (such as those generated by the *Layer Preview* page) since it relaxes the need to click precisely on a point for the appropriate feature info to be returned.

In both operations GeoServer attempts to compute the `buffer` value automatically by inspecting the styles for each layer. All active symbolizers are evaluated, and the size of the largest is used (i.e. largest point symbolizer, thickest line symbolizer). Automatic buffer sizing cannot be computed if:

- the SLD contains sizes that are specified as feature attribute values
- the SLD contains external graphics and does not specify their size explicitly

In this event, the following defaults are used:

- 0 pixels for *GetMap* requests
- 5 pixels for *GetFeatureInfo* requests (a different min value can be set via the `org.geoserver.wms.featureinfo.minBuffer` system variable, e.g., add `-Dorg.geoserver.wms.featureinfo.minBuffer=10` to make the min buffer be 10 pixels)

If these are not sufficiently large, the explicit parameter can be used.

cql_filter

The `cql_filter` parameter is similar to the standard `filter` parameter, but the filter is expressed using ECQL (Extended Common Query Language). ECQL provides a more compact and readable syntax compared to OGC XML filters. For full details see the *ECQL Reference* and *CQL and ECQL* tutorial.

If more than one layer is specified in the `layers` parameter, then a separate filter can be specified for each layer, separated by semicolons. The syntax is:

```
cql_filter=filter1;filter2...
```

An example of a simple CQL filter is:

```
cql_filter=INTERSECTS(the_geom,%20POINT%20(-74.817265%2040.5296504))
```

sortBy

The `sortBy` parameter allows to control the order of features/rasters displayed in the map, using the same syntax as WFS 1.0, that is:

- `&sortBy=att1 A|D,att2 A|D, ...` for a single layer request
- `&sortBy=(att1Layer1 A|D,att2Layer1 A|D, ...)(att1Layer2 A|D,att2Layer2 A|D, ...)...` when requesting multiple layers

Care should be taken when using it as it has different behavior for raster layers, vector layers, and layer groups. In particular:

- For **raster layers**, `sortBy` maps to a "SORTING" read parameter that the reader might expose (image mosaic exposes such parameter).
  In image mosaic, this causes the first granule found in the sorting will display on top, and then the others will follow.
  Thus, to sort a scattered mosaic of satellite images so that the most recent image shows on top, and assuming the time attribute is called `ingestion` in the mosaic index, the specification will be `&sortBy=ingestion D`.
- For **vector layers**, `sortBy` maps to a sort by clause in the vector data source, and then painting happens using the normal "painter model" rules, so the first item returned is painted first, and then all others on top of it.
  Thus, to sort a set of event points so that the most recent event is painted on top, and assuming the attribute is called "date" in the vector layer, the specification will be `&sortBy=date` or `&sortBy=date A` (ascending direction being the default one).
- For **layer groups**, the sort specification is going to be copied over all internal layers, so the spec has to be valid for all of them, or an error will be reported.
  An empty spec can be used for layer groups in this case, for example, `layers=theGroup,theLayer&sortBy=(),(date A)`

env

The `env` parameter defines the set of substitution values that can be used in SLD variable substitution. The syntax is:

```
env=param1:value1;param2:value2;...
```

See *Variable substitution in SLD* for more information.

featureid

The `featureid` parameter filters by feature ID, a unique value given to all features. Multiple features can be selected by separating the featureids by comma, as in this example:

```
featureid=states.1,states.45
```

filter

The WMS specification allows only limited filtering of data. GeoServer enhances the WMS filter capability to match that provided by WFS. The `filter` parameter can specify a list of OGC XML filters. The list is enclosed in parentheses: **( )**. When used in a GET request, the XML tag brackets must be URL-encoded.

If more than one layer is specified in the `layers` parameter then a separate filter can be specified for each layer.

An example of an OGC filter encoded in a GET request is:

```
filter=%3CFilter%20xmlns:gml=%22http://www.opengis.net/gml%22%3E%3CIntersects%3E
↪%3CPropertyName%3Ethe_geom%3C/PropertyName%3E%3Cgml:Point%20srsName=%224326%22%3E
↪%3Cgml:coordinates%3E-74.817265,40.5296504%3C/gml:coordinates%3E%3C/gml:Point%3E%3C/
↪Intersects%3E%3C/Filter%3E
```

format_options

The `format_options` is a container for parameters that are format-specific. The syntax is:

```
format_options=param1:value1;param2:value2;...
```

The supported format options are:

- `antialiasing` (values = `on`, `off`, `text`): controls the use of antialiased rendering in raster output.
- `callback`: specifies the callback function name for the jsonp response format (default is `parseResponse`).
- `dpi`: sets the rendering DPI (dots-per-inch) for raster outputs. The OGC standard output resolution is 90 DPI. If you need to create high resolution images (e.g for printing) it is advisable to request a larger image size and specify a higher DPI. In general, the image size should be increased by a factor equal to `targetDPI/90`, with the target dpi set in the format options. For example, to print a 100x100 image at 300 DPI request a 333x333 image with the DPI value set to 300: `&width=333&height=333&format_options=dpi:300`
- `layout`: specifies a layout name to use. Layouts are used to add decorators such as compasses and legends. This capability is discussed further in the *WMS Decorations* section.
- `quantizer` (values = `octree`, `mediancut`): controls the color quantizer used to produce PNG8 images. GeoServer 2.2.0 provides two quantizers, a fast RGB quantizer called `octree` that does not handle translucency and a slower but more accurate RGBA quantizer called `mediancut`. By default the first is used on opaque images, whilst the second is enabled if the client asks for a transparent image (`transparent=true`). This vendor parameter can be used to manually force the usage of a particular quantizer.
- `timeout`: Apply a timeout value for a getMap request. If the timeout is reached, the getMap request is cancelled and an error is returned. The value used for the timeout will be the minimum of this format option and the global WMS timeout defined in the *WMS configuration*. A value of zero means no timeout.
- `kmattr` (values = `true`, `false`): determines whether the KML returned by GeoServer should include clickable attributes or not. This parameter primarily affects Google Earth rendering.
- `legend` (values = `true`, `false`): KML may add the legend.

- `kmscore` (values = between `0` to force raster output and `100` to force vector output): parameter sets whether GeoServer should render KML data as vector or raster. This parameter primarily affects Google Earth rendering.
- `kmltitle`: parameter sets the KML title.
- `kmlrefresh` (values = greater than `0` or `expires`): Force Network Link reload in refresh mode on interval of seconds. When expires is specified client will refresh whenever the time has elapsed specified in cache expiration headers. The caching time may be set in the Layer configuration under Publishing tab setting HTTP Cache Time. This parameter primarily affects Google Earth rendering and is dependent on being respected by the client. Using a second interval is a more reliable choice.
- `kmlvisible` (values = `true`, `false`): Indicates whether layers selected will default to enabled or not. Default behavior is enabled. This parameter primarily affects Google Earth rendering.
- `advancedProjectionHandling` (values = `true`, `false`): Enable Disable advanced projection handling, if it is enabled in the GUI. If it is disabled in the GUI, this option has no effect.
- `mapWrapping` (values = `true`, `false`): Enable Disable continuous map wrapping, if it is enabled in the GUI. If it is disabled in the GUI, this option has no effect. Continuous map wrapping will also be disabled if `advancedProjectionHandling` is disabled.

maxFeatures and startIndex

The parameters `maxFeatures` and `startIndex` can be used together to provide "paging" support. Paging is helpful in situations such as KML crawling, where it is desirable to be able to retrieve the map in sections when there are a large number of features.

The `startindex=n` parameter specifies the index from which to start rendering in an ordered list of features. `n` must be a positive integer.

The `maxfeatures=n` parameter sets a limit on the amount of features rendered. `n` must be a positive integer. When used with `startindex`, the features rendered will be the ones starting at the `startindex` value.

Note that not all layers support paging. For a layer to be queried in this way, the underlying feature source must support paging. This is usually the case for databases (such as PostGIS).

namespace

The `namespace` parameter causes WMS *GetCapabilities* responses to be filtered to only contain layers in to a particular namespace. The syntax is:

```
namespace=<namespace>
```

where `<namespace>` is the namespace prefix.

Warning: Using an invalid namespace prefix will not cause an error, but the capabilities document returned will contain no layers, only layer groups.

Note: This affects the capabilities document only, not other requests. Other WMS operations will still process all layers, even when a namespace is specified.

palette

It is sometimes advisable (for speed and bandwidth reasons) to downsample the bit depth of returned maps. The way to do this is to create an image with a limited color palette, and save it in the `palettes` directory inside your GeoServer Data Directory. It is then possible to specify the `palette` parameter of the form:

```
palette=<image>
```

where `<image>` is the filename of the palette image (without the extension). To force a web-safe palette, use the syntax `palette=safe`. For more information see the tutorial on *Paletted Images*

propertyName

The `propertyName` parameter specifies which properties are included in the response of the `GetFeatureInfo` operation. The syntax is the same as in the WFS `GetFeature` operation. For a request for a single layer the syntax is:

```
propertyName=name1,...,nameN
```

For multiple layers the syntax is:

```
propertyName=(nameLayer11,...,nameLayer1N)...(name1LayerN,...,nameNLayerN)
```

The nature of the properties depends on the layer type:

- For vector layers the names specify the feature attributes.
- For raster layers the names specify the bands.
- For cascaded WMS layers the names specify the GML properties to be returned by the remote server.

tiled

Meta-tiling prevents issues with duplicated labels when using a tiled client such as OpenLayers. When meta-tiling is used, images are rendered and then split into smaller tiles (by default in a 3x3 pattern) before being served. In order for meta-tiling to work, the tile size *must* be set to 256x256 pixels, and the `tiled` and `tilesorigin` parameters must be specified.

The `tiled` parameter controls whether meta-tiling is used. The syntax is:

```
tiled=[true|false]
```

To invoke meta-tiling use `tiled=true`.

tilesorigin

The `tilesorigin` parameter is also required for meta-tiling. The syntax is:

```
tilesorigin=x,y
```

where `x` and `y` are the coordinates of the lower left corner (the "origin") of the tile grid system.

OpenLayers example

In OpenLayers, a good way to specify the `tilesorigin` is to reference the map extents directly.

> **Warning:** If the map extents are modified dynamically, the `tilesorigin` of each meta-tiled layer must be updated accordingly.

The following code shows how to specify the meta-tiling parameters:

```
var options = {
    ...
    maxExtent: new OpenLayers.Bounds(-180, -90, 180, 90),
    ...
};
map = new OpenLayers.Map('map', options);

tiled = new OpenLayers.Layer.WMS(
    "Layer name", "http://localhost:8080/geoserver/wms",
    {
        srs: 'EPSG:4326',
        width: 391,
        styles: '',
        height: 550,
        layers: 'layerName',
        format: 'image/png',
        tiled: true,
        tilesorigin: map.maxExtent.left + ',' + map.maxExtent.bottom
    },
    {buffer: 0}
);
```

scaleMethod

The `scaleMethod` parameter controls how the scale denominator is computed by GeoServer The two possible values are:

- **`OGC` (default): the scale denominator is computed according to the OGC SLD specification, which** imposes simplified formulas for the sake of interoperability
- **`Accurate`: use the full expressions for computing the scale denominator against geographic** data, taking into account the ellipsoidal shape of Earth

The two methods tend to return values rather close to each other near the equator, but they do diverge to larger differences as the latitude approaches the poles.

interpolations

The `interpolations` parameter allows choosing a specific resampling (interpolation) method. It can be used in the `GetMap` operation.

If more than one layer is specified in the `layers` parameter, then a separate interpolation method can be specified for each layer, separated by commas. The syntax is:

```
interpolations=method1,method2,...
```

method<n> values can be one of the following:

- **nearest neighbor**
- **bilinear**
- **bicubic**

or empty if the default method has to be used for the related layer.

The parameter allows to override the global *WMS Raster Rendering Options* setting (see *WMS Settings* for more info), as well as the layer specific *Default Interpolation Method* publishing parameter (see *Layers* for more info), on a layer by layer basis.

format

The `format` parameter can be used to request capabilities documents in a certain format. If the requested format is not supported the default format will be used.

An example request:

> http://localhost:8080/geoserver/ows?service=wms&version=1.1.1&request=GetCapabilities&format=text/xml

Note: Currently this parameter can only be used to request WMS 1.1.1 capabilities documents encoded in `text/xml`, if used with other WMS versions or other formats it will have no effect.

7.1.7 Non Standard AUTO Namespace

The WMS standard supports a small number of "automatic" coordinate reference systems that include a user-selected centre of projection. These are specified using:

```
AUTO:auto_crs_id,factor,lon0,lat0
```

for example:

```
CRS=AUTO:42003,1,-100,45
```

Note: in GeoServer 2.8.x AUTO and AUTO2 namespaces are treated identically.

Note: in GeoServer 2.8.x the factor parameter in the AUTO namespace is ignored. The BBOX parameter to GetMap must therefore be specified in metres.

The WMS standard provide projections with IDs in the range 42001 to 42005.

ID	Projection
42001	Universal Transverse Mercator
42002	Transverse Mercator
42003	Orthographic
42004	Equirectangular
42005	Mollweide (not supported in GeoServer 2.8.x)

GeoServer also supports some non-standard coordinate reference systems. These are

ID	Projection
97001	Gnomonic
97002	Stereographic

Note: the auto stereographic projection uses a sphere. It does this by setting the semi minor axis to the same value as the semi major axis.

7.1.8 WMS configuration

Layer Groups

A Layer Group is a group of layers that can be referred to by one layer name. For example, if you put three layers (call them layer_A, layer_B, and layer_C) under the one "Layer Group" layer, then when a user makes a WMS getMap request for that group name, they will get a map of those three layers.

For information on configuring Layer Groups in the Web Administration Interface see *Layer Groups*

Request limits

The request limit options allow the administrator to limit the resources consumed by each WMS `GetMap` request.

The following table shows the option names, a description, and the minimum GeoServer version at which the option is available (older versions will ignore it if set).

Option	Description	Version
Max rendering memory (KB)	Sets the maximum amount of memory a single GetMap request is allowed to use (in kilobytes). The limit is checked before request execution by estimating how much memory would be required to produce the output in the format requested. For example, for an image format the estimate is based on the size of the required rendering memory (which is determined by the image size, the pixel bit depth, and the number of active FeatureTypeStyles at the requested scale). If the estimated memory size is below the limit, the request is executed; otherwise it is cancelled.	1.7.5
Max rendering time (s)	Sets the maximum amount of time GeoServer will spend processing a request (in seconds). This time limits the "blind processing" portion of the request, that is, the time taken to read data and compute the output result (which may occur concurrently). If the execution time reaches the limit, the request is cancelled. The time required to write results back to the client is not limited by this parameter, since this is determined by the (unknown) network latency between the server and the client. For example, in the case of PNG/JPEG image generation, this option limits the data reading and rendering time, but not the time taken to write the image out.	1.7.5
Max rendering errors (count)	Sets the maximum amount of rendering errors tolerated by a GetMap request. By default GetMap makes a best-effort attempt to serve the result, ignoring invalid features, reprojection errors and the like. Setting a limit on the number of errors ignored can make it easier to notice issues, and conserves CPU cycles by reducing the errors which must be handled and logged	1.7.5
Max number of dimension values	Sets the maximum number of dimension (time, elevation, custom) values that a client can request in a GetMap/GetFeatureInfo request (the work to be done is usually proportional to said number of times, and the list of values is kept in memory during the processing)	2.14.0

The default value of each limit is `0`, which specifies that the limit is not applied.

If any of the request limits is exceeded, the GetMap operation is cancelled and a `ServiceException` is returned to the client.

When setting the above limits it is suggested that peak conditions be taken into consideration. For example, under normal circumstances a GetMap request may take less than a second. Under high load it is acceptable for it to take longer, but it's usually not desirable to allow a request to go on for 30 minutes.

The following table shows examples of reasonable values for the request limits:

Option	Value	Rationale
maxRequestMemory	16384	16MB are sufficient to render a 2048x2048 image at 4 bytes per pixel (full color and transparency), or a 8x8 meta-tile when using GeoWebCache or TileCache. Note that the rendering process uses a separate memory buffer for each FeatureTypeStyle in an SLD, so this also affects the maximum image size. For example, if an SLD contains two FeatureTypeStyle elements in order to draw cased lines for a highway, the maximum image size will be limited to 1448x1448 (the memory requirement increases with the product of the image dimensions, so halving the memory decreases image dimensions by only about 30%)
maxRenderingTime	120	A request that processes for a full two minutes is probably rendering too many features, regardless of the current server load. This may be caused by a request against a big layer using a style that does not have suitable scale dependencies
maxRenderingErrors	100	Encountering 100 errors is probably the result of a request trying to reproject a big data set into a projection that is not appropriate for the output extent, resulting in many reprojection failures.

7.1.9 Global variables affecting WMS

This document details the set of global variables that can affect WMS behaviour. Each global variable can be set as an environment variable, as a servlet context variable, or as a Java system property, just like the well known `GEOSERVER_DATA_DIR` setting. Refer to *Setting the data directory location* for details on how a global variable can be specified.

MAX_FILTER_RULES

A integer number (defaults to 20) When drawing a style containing multiple active rules the renderer combines the filters of the rules in OR and adds them to the standard bounding box filter. This behaviour is active up until the maximum number of filter rules is reached, past that the rule filters are no more added to avoid huge queries. By default up to 20 rules are combined, past 20 rules only the bounding box filter is used. Turning it off (setting it to 0) can be useful if the styles are mostly classifications, detrimental if the rule filters are actually filtering a good amount of data out.

OPTIMIZE_LINE_WIDTH

Can be `true` or `false` (defaults to: `false`). When `true` any stroke whose width is less than 1.5 pixels gets slimmed down to "zero", which is actually not zero, but a very thin line. That was the behaviour GeoServer used to default to before the 2.0 series. When `false` the stroke width is not modified and it's possible to specify widths less than one pixel. This is the default behaviour starting from the 2.0.0 release

ENABLE_JSONP

Can be `true` or `false` (defaults to: `false`). When `true` the JSONP (text/javascript) output format is enabled.

7.1.10 GetLegendGraphic

This chapter describes whether to use the GetLegendGraphics request. The SLD Specifications 1.0.0 gives a good description about GetLegendGraphic requests:

The GetLegendGraphic operation itself is optional for an SLD-enabled WMS. It provides a general mechanism for acquiring legend symbols, beyond the LegendURL reference of WMS Capabilities. Servers supporting the GetLegendGraphic call might code LegendURL references as GetLegendGraphic for interface consistency. Vendor-specific parameters may be added to GetLegendGraphic requests and all of the usual OGC-interface options and rules apply. No XML-POST method for GetLegendGraphic is presently defined.

Here is an example invocation:

```
http://localhost:8080/geoserver/wms?REQUEST=GetLegendGraphic&VERSION=1.0.0&
↪FORMAT=image/png&WIDTH=20&HEIGHT=20&LAYER=topp:states
```

which would produce four 20x20 icons that graphically represent the rules of the default style of the topp:states layer.

Fig. 7.5: *Sample legend*

In the following table the whole set of GetLegendGraphic parameters that can be used.

Parameter	Required	Description
REQUEST	Required	Value must be "GetLegendGraphic".
LAYER	Required	Layer for which to produce legend graphic.
STYLE	Optional	Style of layer for which to produce legend graphic. If not present, the default style is selected. The style may be any valid style available for a layer, including non-SLD internally-defined styles.
FEATURETYPE	Optional	Feature type for which to produce the legend graphic. This is not needed if the layer has only a single feature type.
RULE	Optional	Rule of style to produce legend graphic for, if applicable. In the case that a style has multiple rules but no specific rule is selected, then the map server is obligated to produce a graphic that is representative of all of the rules of the style.
SCALE	Optional	In the case that a RULE is not specified for a style, this parameter may assist the server in selecting a more appropriate representative graphic by eliminating internal rules that are out-of-scope. This value is a standardized scale denominator, defined in Section 10.2. Specifying the scale will also make the symbolizers using Unit Of Measure resize according to the specified scale.
SLD	Optional	This parameter specifies a reference to an external SLD document. It works in the same way as the SLD= parameter of the WMS GetMap operation.
SLD_BODY	Optional	This parameter allows an SLD document to be included directly in an HTTP-GET request. It works in the same way as the SLD_BODY= parameter of the WMS GetMap operation.
FORMAT	Required	This gives the MIME type of the file format in which to return the legend graphic. Allowed values are the same as for the FORMAT= parameter of the WMS GetMap request.
WIDTH	Optional	This gives a hint for the width of the returned graphic in pixels. Vector-graphics can use this value as a hint for the level of detail to include.
HEIGHT	Optional	This gives a hint for the height of the returned graphic in pixels.
EXCEPTIONS	Optional	This gives the MIME type of the format in which to return exceptions. Allowed values are the same as for the EXCEPTIONS= parameter of the WMS GetMap request.
LANGUAGE	Optional	Allows setting labels language for style titles and rules titles; needs a correctly localized SLD to work properly; if labels are not available in the requested language, the default text will be used; look at *i18N in SLD* for further details.

Controlling legend appearance with LEGEND_OPTIONS

GeoServer allows finer control over the legend appearance via the vendor parameter `LEGEND_OPTIONS`. The general format of `LEGEND_OPTIONS` is the same as `FORMAT_OPTIONS`, that is:

```
...&LEGEND_OPTIONS=key1:v1;key2:v2;...;keyn:vn
```

Here is a description of the various parameters that can be used in `LEGEND_OPTIONS`:

- **fontName (string)** the name of the font to be used when generating rule titles. The font must be available on the server
- **fontStyle (string)** can be set to italic or bold to control the text style. Other combination are not allowed right now but we could implement that as well.
- **fontSize (integer)** allows us to set the Font size for the various text elements. Notice that default size is 12.
- **fontColor (hex)** allows us to set the color for the text of rules and labels (see above for recommendation on how to create values). Values are expressed in `0xRRGGBB` format

- **fontAntiAliasing (true/false)** when true enables antialiasing for rule titles
- **bgColor (hex)** background color for the generated legend, values are expressed in `0xRRGGBB` format
- **dpi (integer)** sets the DPI for the current request, in the same way as it is supported by GetMap. Setting a DPI larger than 91 (the default) makes all fonts, symbols and line widths grow without changing the current scale, making it possible to get a high resolution version of the legend suitable for inclusion in printouts
- **forceLabels** "on" means labels will always be drawn, even if only one rule is available. "off" means labels will never be drawn, even if multiple rules are available. Off by default.
- **labelMargin** margin (in pixels) to use between icons and labels.
- **layout** sets icons layout to be **vertical** (default) or **horizontal**.
- **columnheight** enables **multicolumn** layout when layout is **vertical**. Each column height is limited by the columnheight value (in pixels).
- **rowwidth** enables **multirow** layout when layout is **horizontal**. Each row width is limited by the rowwidth value (in pixels).
- **columns** enables **multicolumn** layout when layout is **vertical**. The value is the maximum columns number of legend. The rows used are equal to the next greater integer of <total of icons>/<number of columns>.
- **rows** enables **multirow** layout when layout is **horizontal**. The value is the the maximum rows number of legend. The columns used are equal to the next greater integer of <total of icons>/<number of rows>.
- **grouplayout** Orientation of groups of layer, possible values are **horizontal** and **vertical** (default if not specified).
- **countMatched** When set to true, adds at the end of each label the number of features matching that rule in the current map. Requires extra parameters, see details in the *dedicated section*.

Here is a sample request sporting most the options:

```
http://localhost:8080/geoserver/wms?REQUEST=GetLegendGraphic&VERSION=1.0.0&
↪FORMAT=image/png&WIDTH=20&HEIGHT=20&LAYER=topp:states&legend_options=fontName:Times
↪%20New%20Roman;fontAntiAliasing:true;fontColor:0x000033;fontSize:14;
↪bgColor:0xFFFFEE;dpi:180
```

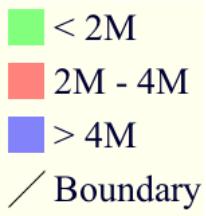


Fig. 7.6: *Using LEGEND_OPTIONS to control the output*

Controlling legend layout

A set of LEGEND_OPTIONS keys are used to control icons layout in the produced legend images. In particular, a **vertical** or **horizontal** layout can be chosen.

Multi column or multi row layouts are possible, and are controlled by the columnheight / rowwidth options (to limit each column / row size) or by the columns / rows options (to fix the # of columns / rows to be used).

Both columnheight / columns and rowwidth / rows can be used to limit the whole size of the produced image (some icons are skipped id they do not fit into the given limits).

In addition, orientation of legends in a layergroup can be configured using the grouplayout option.

Raster Legends Explained

This chapter aim to briefly describe the work that I have performed in order to support legends for raster data that draw information taken from the various bits of the SLD 1.0 RasterSymbolizer element. Recall, that up to now there was no way to create legends for raster data, therefore we have tried to fill the gap by providing an implementation of the getLegendGraphic request that would work with the ColorMap element of the SLD 1.0 RasterSymbolizer. Notice that some "debug" info about the style, like colormap type and band used are printed out as well.

What's a raster legend

Here below I have drawn the structure of a typical legend, where some elements of interests are parameterized.

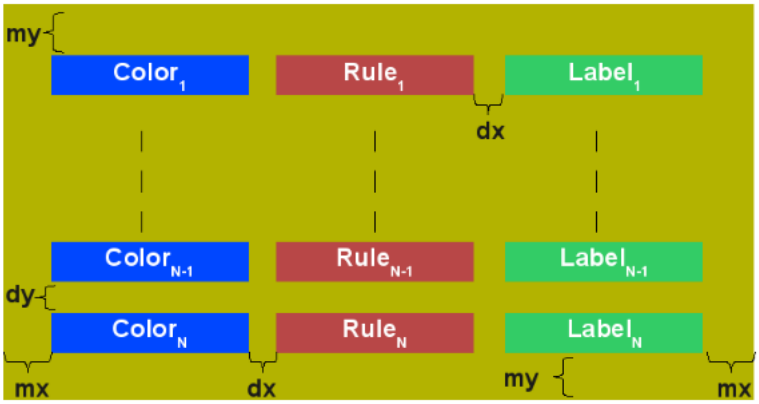


Fig. 7.7: *The structure of a typical legend*

Take as an instance one of the SLD files attached to this page, each row in the above table draws its essence from the ColorMapEntry element as shown here below:

```
<ColorMapEntry color="#732600" quantity="9888" opacity="1.0" label="<-70 mm"/>
```

The producer for the raster legend will make use of this elements in order to build the legend, with this regards, notice that:

- the width of the Color element is driven by the requested width for the GetLegendGraphic request
- the width and height of label and rules is computed accordingly to the used Font and Font size for the prepared text (**no new line management for the moment**)
- the height of the Color element is driven by the requested width for the GetLegendGraphic request, but notice that for ramps we expand this a little since the goal is to turn the various Color elements into a single long strip

- the height of each row is set to the maximum height of the single elements
- the width of each row is set to the sum of the width of the various elements plus the various paddings
- **dx,dy** the spaces between elements and rows are set to the 15% of the requested width and height. Notice that **dy** is ignored for the colormaps of type **ramp** since they must create a continuous color strip.
- **absoluteMargins** true/false, used to change the uom of **dx** from percentage (when false) to a fixed number of pixels (when true).
- **mx,my** the margins from the border of the legends are set to the 1.5% of the total size of the legend

Just to jump right to the conclusions (which is a bad practice I know, but no one is perfect), here below I am adding an image of a sample legend with all the various options at work. The request that generated it is the following:

```
http://localhost:8081/geoserver/wms?REQUEST=GetLegendGraphic&VERSION=1.0.0&
↪FORMAT=image/png&WIDTH=100&HEIGHT=20&LAYER=it.geosolutions:di08031_da&LEGEND_
↪OPTIONS=forceRule:True;dx:0.2;dy:0.2;mx:0.2;my:0.2;fontStyle:bold;
↪borderColor:0000ff;border:true;fontColor:ff0000;fontSize:18
```

Do not worry if it seems like something written in ancient dead language, I am going to explain the various params here below.

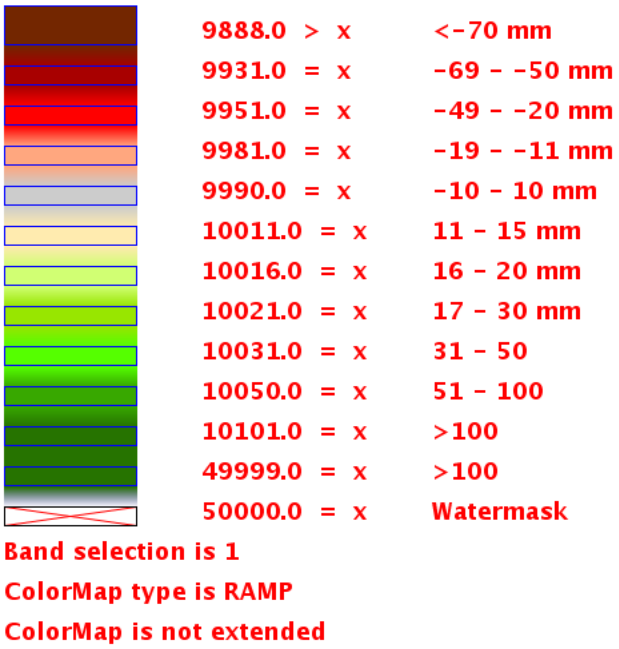


Fig. 7.8: *Example of a raster legend*

Raster legends' types

As you may know (well, actually you might not since I never wrote any real docs about the RasterSymbolizer work I did) GeoServer supports three types of ColorMaps:

- **ramp** this is what SLD 1.0 dictates, which means a linear interpolation weighted on values between the colors of the various ColorMapEntries.

- **values** this is an extensions that allows link quantities to colors as specified by the ColorMapEntries quantities. Values not specified are translated into transparent pixels.
- **classes** this is an extensions that allows pure classifications based o intervals created from the ColorMapEntries quantities. Values not specified are translated into transparent pixels.

Here below I am going to list various examples that use the attached styles on a rainfall floating point geotiff.

ColorMap type is VALUES

Refer to the SLD rainfall.sld in attachment.

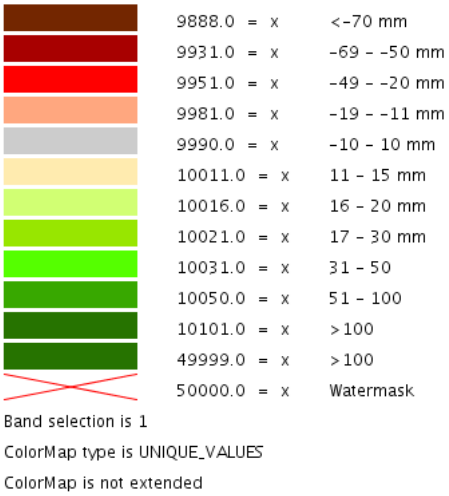


Fig. 7.9: *Raster legend - VALUES type*

ColorMap type is CLASSES

Refer to the SLD rainfall_classes.sld in attachment.

ColorMap type is RAMP

Refer to the SLD rainfall_classes.sld in attachment. Notice that the first legend show the default border behavior while the second has been force to draw a border for the breakpoint color of the the colormap entry quantity described by the rendered text. Notice that each color element has a part that show the fixed color from the colormap entry it depicts (the lowest part of it, the one that has been outlined by the border in the second legend below) while the upper part of the element has a gradient that connects each element to the previous one to point out the fact that we are using linear interpolation.

The various control parameters and how to set them

I am now going to briefly explain the various parameters that we can use to control the layout and content of the legend. A request that puts all the various options is shown here:

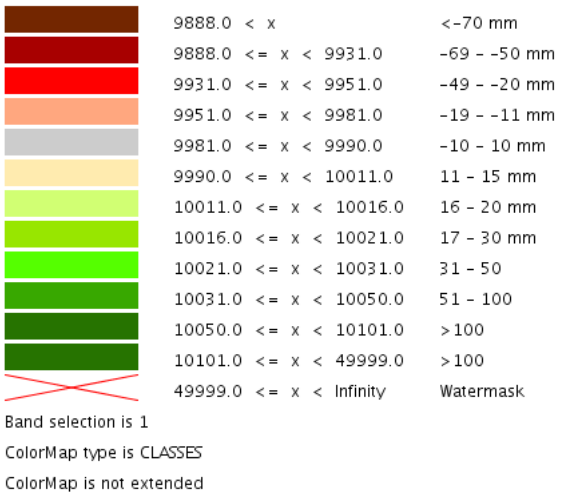


Fig. 7.10: *Raster legend - CLASSES type*

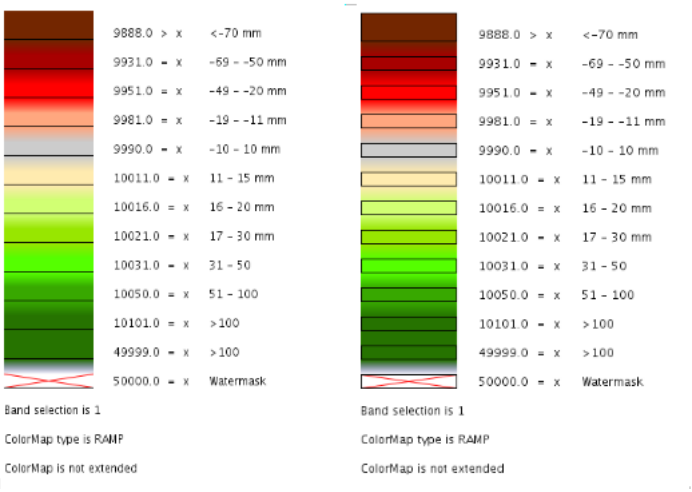


Fig. 7.11: *Raster legend - RAMP type*

```
http://localhost:8081/geoserver/wms?REQUEST=GetLegendGraphic&VERSION=1.0.0&
↪FORMAT=image/png&WIDTH=100&HEIGHT=20&LAYER=it.geosolutions:di08031_da&LEGEND_
↪OPTIONS=forceRule:True;dx:0.2;dy:0.2;mx:0.2;my:0.2;fontStyle:bold;
↪borderColor:0000ff;border:true;fontColor:ff0000;fontSize:18
```

Let's now examine all the interesting elements, one by one. Notice that I am not going to discuss the mechanics of the GetLegendGraphic operation, for that you may want to refer to the SLD 1.0 spec, my goal is to briefly discuss the LEGEND_OPTIONS parameter.

- **forceRule (boolean)** by default rules for a ColorMapEntry are not drawn to keep the legend small and compact, unless there are no labels at all. You can change this behaviour by setting this parameter to true.
- **dx,dy,mx,my (double)** can be used to set the margin and the buffers between elements
- **border (boolean)** activates or deactivates the border on the color elements in order to make the separations clearer. Notice that I decided to **always** have a line that would split the various color elements for the ramp type of colormap.
- **borderColor (hex)** allows us to set the color for the border in 0xRRGGBB format

CQL Expressions and ENV

If cql expressions are used in ColorMapEntry attributes (see *here*) to create a dynamic color map taking values from ENV, the same ENV parameters used for GetMap can be given to GetLegendGraphic to get the desired legend entries.

Content dependent legends

GeoServer allows building content dependent legend, that is, legends whose contents depend on the currently displayed map. In order to support it the GetLegendGraphic call needs the following extra parameters:

- BBOX
- SRS or CRS (depending on the WMS version, SRS for 1.1.1 and CRS for 1.3.0)
- SRCWITH and SRCHEIGHT, the size of the reference map (width and height already have a different meaning in GetLegendGraphic)

Other parameters can also be added to better match the GetMap request, for example, it is recommended to mirror filtering vendor parameters such as, for example, CQL_FILTER,FILTER,FEATUREID,TIME,ELEVATION.

Content dependent evaluation is enabled via the following LEGEND_OPTIONS parameters:

- countMatched: adds the number of features matching the particular rule at the end of the rule label (requires visible labels to work). Applicable only to vector layers.

More approaches may be added in future (e.g., making rules that are not matching any feature disappear).

For example, let's assume the following layout is added to GeoServer (`legend.xml` to be placed in `GEOSERVER_DATA_DIR/layouts`):

```
<layout>
    <decoration type="legend" affinity="top,right" offset="0,0" size="auto"/>
</layout>
```

This will make a legend appear in the GetMap response. The following preview request uses the layout to embed a legend and activates feature counting in it:

```
http://localhost:8080/geoserver/topp/wms?service=WMS&version=1.1.0&request=GetMap&
↪layers=topp:states&styles=&bbox=-124.73142200000001,24.955967,-66.969849,49.371735&
↪width=768&height=330&srs=EPSG:4326&format=application/openlayers&format_
↪options=layout:legend&legend_options=countMatched:true;fontAntiAliasing:true
```

The result will look as follows:

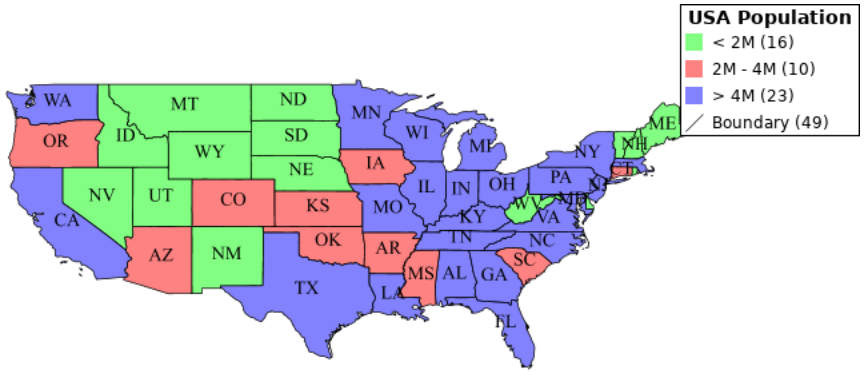


Fig. 7.12: *Embedded legend, full map*

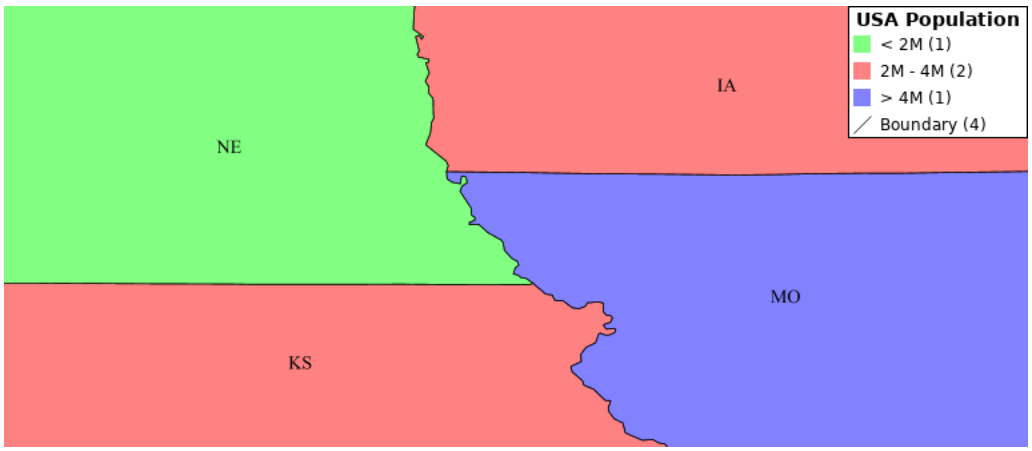


Fig. 7.13: *Embedded legend, four states*

The same can be achieved using a stand-alone GetLegendGraphic request:

```
http://localhost:8080/geoserver/topp/wms?service=WMS&version=1.1.0&
↪request=GetLegendGraphic&width=20&height=20&layer=topp:states&bbox=-124.
↪73142200000001,24.955967,-66.969849,49.371735&srcwidth=768&srcheight=330&
↪srs=EPSG:4326&format=image/png&legend_options=countMatched:true;
↪fontAntiAliasing:true
```

Fig. 7.14: *Embedded legend, single state*

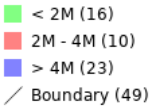


Fig. 7.15: *Direct legend request*

JSON Output Format

Since version 2.15.0 it has been possible to use **application/json** as an output format in GetLegendGraphic requests. This allows a JSON aware client to receive a JSON representation of the legend graphic to use for its own rendering requirements.

A simple http request can be used:

```
http://localhost:9000/geoserver/wms?service=WMS&version=1.1.0&
↪request=GetLegendGraphic&layer=topp:states&format=application/json
```

Which returns a JSON response:

```
{"Legend": [{
  "layerName": "states",
  "title": "USA Population",
  "rules":   [
        {
      "title": "< 2M",
      "filter": "[PERSONS < '2000000']",
      "symbolizers": [{"Polygon":       {
        "fill": "#4DFF4D",
        "fill-opacity": "0.7"
      }}]
    },
        {
      "title": "2M - 4M",
      "filter": "[PERSONS BETWEEN '2000000' AND '4000000']",
      "symbolizers": [{"Polygon":       {
        "fill": "#FF4D4D",
        "fill-opacity": "0.7"
      }}]
```

```
    },
        {
      "title": "> 4M",
      "filter": "[PERSONS > '4000000']",
      "symbolizers": [{"Polygon":       {
        "fill": "#4D4DFF",
        "fill-opacity": "0.7"
      }}]
    },
        {
      "title": "Boundary",
      "symbolizers":       [
        {"Line":         {
          "stroke": "#000000",
          "stroke-width": "0.2",
          "stroke-opacity": "1",
          "stroke-linecap": "butt",
          "stroke-linejoin": "miter"
        }},
        {"Text":         {
          "label": "[STATE_ABBR]",
          "fonts": [          {
            "font-family": ["Times New Roman"],
            "font-style": "Normal",
            "font-weight": "normal",
            "font-size": "14"
          }],
          "label-placement":           {
            "x-anchor": "0.5",
            "y-anchor": "0.5",
            "rotation": "0.0"
          }
        }}
      ]
    }
  ]
}]}
```

This JSON contains an array of Legends (one for each layer requested) and each legend contains some metadata about the legend and an array of `rule` objects for each rule with in the feature type of the style. Each `rule` contains the metadata associated with the rule, any `filter` element and an array of `symbolizer` objects.

Filters and Expressions

Filters are encoded using *ECQL*, a rule with an ElseFilter has an element "ElseFilter" set to the value "true". Expressions are also encoded in ECQL (wrapped in []) when encountered in the style.

Symbolizers

- PointSymbolizer

  A point symbolizer will be represented as a series of elements containing metadata and an array of `graphics` symbols (see *here*) , these can be well known `marks` or external graphics. The point

symbolizer also provides an "url" element which allows a client to make a request back to GeoServer to fetch a png image of the point symbol.

```
{"Point":     {
    "title": "title",
    "abstract": "abstract",
    "url": "http://localhost:9000/geoserver/kml/icon/capitals?0.0.0=",
    "size": "6",
    "opacity": "1.0",
    "rotation": "0.0",
    "graphics": [      {
      "mark": "circle",
      "fill": "#FFFFFF",
      "fill-opacity": "1.0",
      "stroke": "#000000",
      "stroke-width": "2",
      "stroke-opacity": "1",
      "stroke-linecap": "butt",
      "stroke-linejoin": "miter"
    }]}}
```

- LineSymbolizer

  A line symbolizer is represented as a list of metadata elements and the *stroke parameters*, it is possible for there to be a *graphic-stroke* element too.

```
{"Line":     {
  "stroke": "#AA3333",
  "stroke-width": "2",
  "stroke-opacity": "1",
  "stroke-linecap": "butt",
  "stroke-linejoin": "miter"
}}


{"Line":       {
    "graphic-stroke":         {
      "url": "http://local-test:8080/geoserver/kml/icon/Default Styler",
      "size": "6",
      "opacity": "0.4",
      "rotation": "[(rotation*-1)]",
      "graphics": [          {
        "mark": "square",
        "fill": "#FFFF00",
        "fill-opacity": "1.0"
      }]
    },
    "stroke-opacity": "1",
    "stroke-linecap": "butt",
    "stroke-linejoin": "miter",
    "perpendicular-offset": "10"
  }}
```

- PolygonSymbolizer

  A polygon symbolizer contains *stroke parameters* and *fill parameters*.

```
{"Polygon":       {
  "stroke": "#000000",
```

```
  "stroke-width": "0.5",
  "stroke-opacity": "1",
  "stroke-linecap": "butt",
  "stroke-linejoin": "miter",
  "fill": "#0099CC",
  "fill-opacity": "1.0"
}}
```

Or a graphic stroke and/or a graphic fill (as described above).

```
{"Polygon":       {
  "graphic-stroke":         {
    "url": "http://local-test:8080/geoserver/kml/icon/Default Styler",
    "size": "6",
    "opacity": "0.4",
    "rotation": "[(rotation*-1)]",
    "graphics": [          {
      "mark": "square",
      "fill": "#FFFF00",
      "fill-opacity": "1.0"
    }]
  },
  "stroke-opacity": "1",
  "stroke-linecap": "butt",
  "stroke-linejoin": "miter",
  "graphic-fill":         {
    "url": "http://local-test:8080/geoserver/kml/icon/Default Styler",
    "size": "4",
    "opacity": "0.4",
    "rotation": "[(rotation*-1)]",
    "graphics": [          {
      "mark": "circle",
      "fill": "#FFFFFF",
      "fill-opacity": "1.0"
    }]
  }}}
```

- RasterSymbolizer

  Raster symbolizers contain a `colormap` with an array of `entries`, each entry contains a `label`, `quantity` and `color` element.

```
{"Raster":   {
  "colormap":     {
    "entries":       [
              {
        "label": "values",
        "quantity": "0",
        "color": "#AAFFAA"
      },
              {
        "quantity": "1000",
        "color": "#00FF00"
      },
              {
        "label": "values",
        "quantity": "1200",
        "color": "#FFFF00"
```

```
      },
              {
        "label": "values",
        "quantity": "1400",
        "color": "#FF7F00"
      },
              {
        "label": "values",
        "quantity": "1600",
        "color": "#BF7F3F"
      },
              {
        "label": "values",
        "quantity": "2000",
        "color": "#000000"
      }
    ],
    "type": "ramp"
  },
  "opacity": "1.0"
}}
```

- TextSymbolizer

  A text symbolizer contains a `label` expression, followed by an array of `fonts` and a `label-placement` object containing details of how the label is placed.

```
{"Text":         {
        "label": "[STATE_ABBR]",
        "fonts": [          {
                "font-family": ["Times New Roman"],
                "font-style": "Normal",
                "font-weight": "normal",
                "font-size": "14"
        }],
        "label-placement":           {
                "x-anchor": "0.5",
                "y-anchor": "0.5",
                "rotation": "0.0"
        }
}}
```

Vendor Options

In any case where one or more vendor options is included in the symbolizer there will be a `vendor-options` element included in the output. This object will include one line for each vendor option.

```
"vendor-options": {
  "labelAllGroup": "true",
  "spaceAround": "10",
  "followLine": "true",
  "autoWrap": "50"
}
```

7.1.11 WMS Decorations

WMS Decorations provide a framework for visually annotating images from WMS with absolute, rather than spatial, positioning. Examples of decorations include compasses, legends, and watermarks.

Configuration

To use decorations in a *GetMap* request, the administrator must first configure a decoration layout. These layouts are stored in a subdirectory called `layouts` in the *GeoServer data directory* as XML files, one file per layout. Each layout file must have the extension `.xml`. Once a layout `foo.xml` is defined, users can request it by adding `&format_options=layout:foo` to the request parameters.

Layout files follow a very simple XML structure; a root node named layout containing any number of decoration elements. The order of the decoration elements is the order they are drawn so, in case they are overlapping, the first one will appear under the others.

Each decoration element has several attributes:

Attribute	Meaning
`type`	the type of decoration to use (see *Decoration Types*)
`affinity`	the region of the map image to which the decoration is anchored
`offset`	how far from the anchor point the decoration is drawn
`size`	the maximum size to render the decoration. Note that some decorations may dynamically resize themselves.

Each decoration element may also contain an arbitrary number of option elements providing a parameter name and value:

```
<option name="foo" value="bar"/>
```

Option interpretation depends on the type of decoration in use.

Decoration Types

While GeoServer allows for decorations to be added via extension, there is a core set of decorations included in the default installation. These decorations include:

The **image** decoration (`type="image"`) overlays a static image file onto the document. If height and width are specified, the image will be scaled to fit, otherwise the image is displayed at full size.

Option Name	Meaning
`url`	provides the URL or file path to the image to draw (relative to the GeoServer data directory)
`opacity`	a number from 0 to 100 indicating how opaque the image should be.

The **scaleratio** decoration (`type="scaleratio"`) overlays a text description of the map's scale ratio onto the document.

Option Name	Meaning
`bgcolor`	the background color for the text. supports RGB or RGBA colors specified as hex values.
`fgcolor`	the color for the text and border. follows the color specification from bgcolor.
`format`	the number format pattern, specified using Java own DecimalFormat syntax
`formatLanguage`	the language used to drive number formatting (applies only if also `format` is used), using a valid Java Locale

The **scaleline** decoration (`type="scaleline"`) overlays a graphic showing the scale of the map in world units.

Option Name	Meaning
`bgcolor`	the background color, as used in scaleratio
`fgcolor`	the foreground color, as used in scaleratio
`fontsize`	the size of the font to use
`transparent`	if set to true, the background and border won't be painted (false by default)
`measurement-system`	can be set to "metric" to only show metric units, "imperial" to only show imperial units, or "both" to show both of them (default)

The **legend** decoration (`type="legend"`) overlays a graphic containing legends for the layers in the map.

The **text** decoration (`type="text"`) overlays a parametric, single line text message on top of the map. The parameter values can be fed via the the `env` request parameter, just like SLD enviroment parameters.

Option Name	Meaning
`message`	the message to be displayed, as plain text or Freemarker template that can use the `env` map contents to expand variables
`font-family`	the name of the font used to display the message, e.g., `Arial`, defaults to `Serif`
`font-size`	the size of the font to use (can have decimals), defaults to 12
`font-italic`	it `true` the font will be italic, defaults to `false`
`font-bold`	if `true` the font will be bold, defaults to `false`
`font-color`	the color of the message, in `#RRGGBB` or `#RRGGBBAA` format, defaults to black
`halo-radius`	the radius of a halo around the message, can have decimals, defaults to 0
`halo-color`	the halo fill color, in `#RRGGBB` or `#RRGGBBAA` format, defaults to white

Example

A layout configuration file might look like this:

```
<layout>
    <decoration type="image" affinity="bottom,right" offset="6,6" size="80,31">
        <option name="url" value="pbGS_80x31glow.png"/>
    </decoration>

    <decoration type="scaleline" affinity="bottom,left" offset="36,6"/>

    <decoration type="legend" affinity="top,left" offset="6,6" size="auto"/>
</layout>
```

Used against the states layer from the default GeoServer data, this layout produces an image like the following.

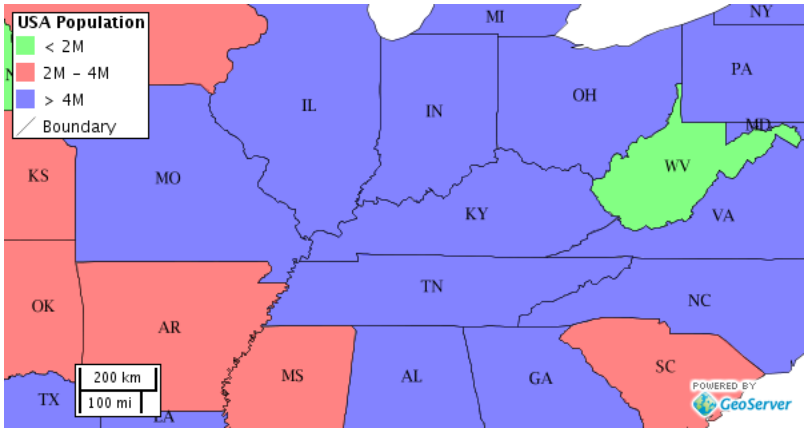


Fig. 7.16: The default states layer, drawn with the decoration layout above.

7.1.12 Google Earth

This section contains information on Google Earth support in GeoServer.

Google Earth is a 3-D virtual globe program. A free download from Google, it allows the user to virtually view, pan, and fly around Earth imagery. The imagery on Google Earth is obtained from a variety of sources, mainly from commercial satellite and aerial photography providers.

Google Earth recognizes a markup language called KML (Keyhole Markup Language) for data exchange. GeoServer integrates with Google Earth by supporting KML as a native output format. Any data configured to be served by GeoServer is thus able to take advantage of the full visualization capabilities of Google Earth.

Overview

Why use GeoServer with Google Earth?

GeoServer is useful when one wants to put a lot of data on to Google Earth. GeoServer automatically generates KML that can be easily and quickly served and visualized in Google Earth. GeoServer operates entirely through a Network Link, which allows it to selectively return information for the area being viewed. With GeoServer as a robust and powerful server and Google Earth providing rich visualizations, they are a perfect match for sharing your data.

Standards-based implementation

GeoServer supports Google Earth by providing KML as a Web Map Service (WMS) output format. This means that adding data published by GeoServer is as simple as constructing a standard WMS request and specifying "**application/vnd.google-earth.kml+xml**" as the `outputFormat`. Since generating KML is just a WMS request, it fully supports *Styling* via SLD.

See the next section (*Quickstart*) to view GeoServer and Google Earth in action.

Quickstart

Note: If you are using GeoServer locally, the GEOSERVER_URL is usually `http://localhost:8080/geoserver`

Viewing a layer

Once GeoServer is installed and running, open up a web browser and go to the web admin console (*Web administration interface*). Navigate to the *Layer Preview* by clicking on the Layer Preview link at the bottom of the left sidebar. You will be presented with a list of the currently configured layers in your GeoServer instance. Find the row that says `topp:states`. To the right of the layer click on the link that says **KML**.

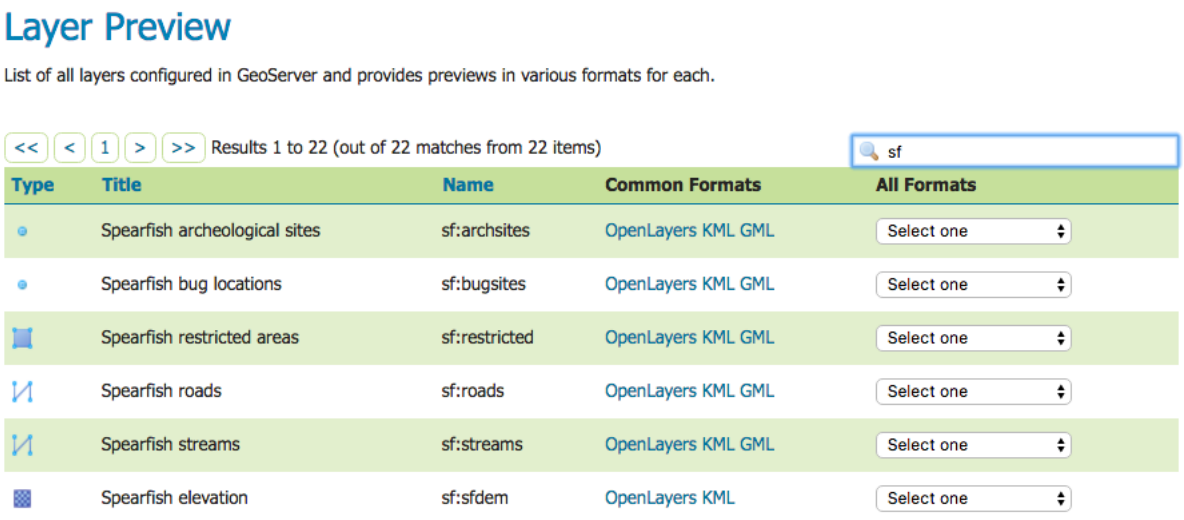


Fig. 7.17: The Map Preview page

If Google Earth is correctly installed on your computer, you will see a dialog asking how to open the file. Select **Open with Google Earth**.

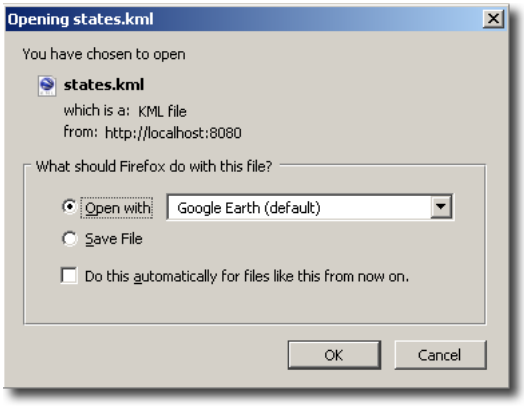


Fig. 7.18: Open with Google Earth

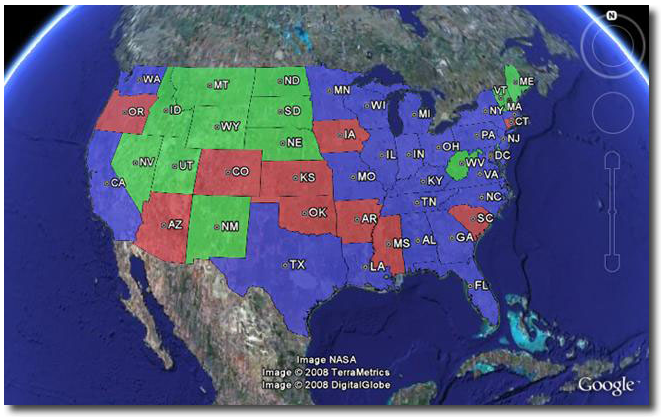


Fig. 7.19: The topp:states layer rendered in Google Earth

When Google Earth is finished loading the result will be similar to below.

Direct access to KML

All of the configured FeatureTypes are available to be output as KML (and thus loaded into Google Earth). The URL structure for KMLs is:

```
http://GEOSERVER_URL/wms/kml?layers=<layername>
```

For example, the topp:states layer URL is:

```
http://GEOSERVER_URL/wms/kml?layers=topp:states
```

Adding a Network Link

An alternative to serving KML directly into Google Earth is to use a Network Link. A Network Link allows for better integration into Google Earth. For example, using a Network Link enables the user to refresh the data within Google Earth, without having to retype a URL, or click on links in the GeoServer Map Preview again.

To add a Network Link, pull down the **Add** menu, and go to **Network Link**. The **New Network Link** dialog box will appear. Name your layer in the **Name** field. (This will show up in **My Places** on the main Google Earth screen.) Set **Link** to:

```
http://GEOSERVER_URL/wms/kml?layers=topp:states
```

(Don't forget to replace the GEOSERVER_URL.) Click **OK**. You can now save this layer in your **My Places**.

Check out the sections on *Tutorials* and the *KML Styling* for more information.

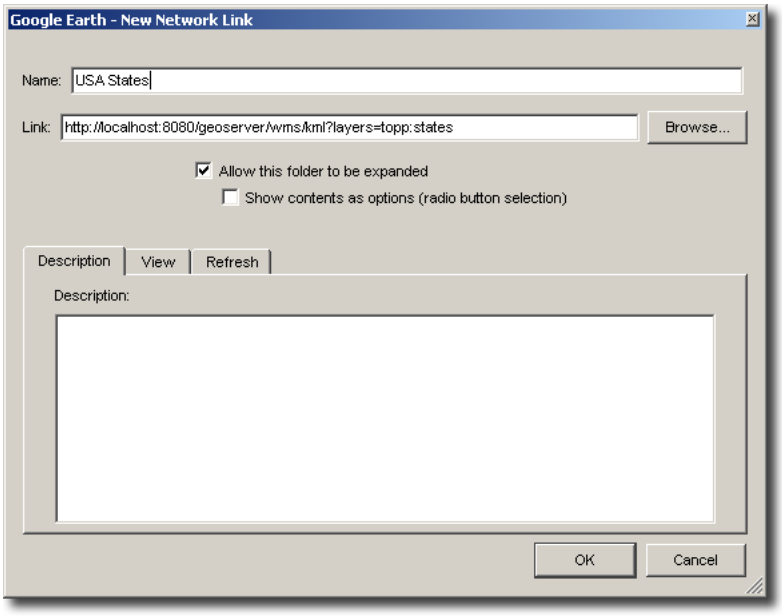


Fig. 7.20: Adding a network link

KML Styling

Introduction

Keyhole Markup Langauge (KML), when created and output by GeoServer, is styled using Styled Layer Descriptors (SLD). This is the same approach used to style standard WMS output formats, but is a bit different from how Google Earth is normally styled, behaving more like Cascading Style Sheets (CSS). The style of the map is specified in the SLD file as a series of rules, and then the data matching those rules is styled appropriately in the KML output. For those unfamiliar with SLD, a good place to start is the *Introduction to SLD*. The remainder of this guide contains information about how to construct SLD documents in order to impact the look of KML produced by GeoServer.

Contents

SLD Generation from CSS

The CSS extension provides the facility to generate SLD files using a lightweight "Cascading Style Sheet" syntax. The CSS GUI provides a live map preview as you are editing your style in addition to an attribute reference for the current layer.

The generated styles will work seamlessly with KML output from GeoServer.

Creating SLD by hand

One can edit the SLD files directly instead of using the CSS extension. For the most complete exploration of editing SLDs see the *Styling* section. The examples below show how some of the basic styles show up in Google Earth.

SLD Structure

The following is a skeleton of a SLD document. It can be used as a base on which to expand upon to create more interesting and complicated styles.

```
<StyledLayerDescriptor version="1.0.0"
   xsi:schemaLocation="http://www.opengis.net/sld StyledLayerDescriptor.xsd"
   xmlns="http://www.opengis.net/sld"
   xmlns:ogc="http://www.opengis.net/ogc"
   xmlns:xlink="http://www.w3.org/1999/xlink"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <NamedLayer>
      <Name>Default Line</Name>
      <UserStyle>
         <Title>My Style</Title>
         <Abstract>A style</Abstract>
         <FeatureTypeStyle>
            <Rule>

                  <!-- symbolizers go here -->

            </Rule>
         </FeatureTypeStyle>
      </UserStyle>
   </NamedLayer>
</StyledLayerDescriptor>
```

Figure 3: Basic SLD structure

In order to test the code snippets in this document, create an SLD with the content as shown in Figure 3, and then add the specific code you wish to test in the space that says `<!-- symbolizers go here -->`. To view, edit, or add SLD files to GeoServer, navigate to **Config** -> **Data** -> **Styles**.

Points

In SLD, styles for points are specified via a PointSymbolizer. An empty PointSymbolizer element will result in a default KML style:

```
<PointSymbolizer>
</PointSymbolizer>
```

Fig. 7.21: *Figure 4: Default point*

Three aspects of points that can be specified are *color*, *opacity*, and the *icon*.

Point Color

The color of a point is specified with a `CssParameter` element and a `fill` attribute. The color is specified as a six digit hexadecimal code.

```
<PointSymbolizer>
   <Graphic>
      <Mark>
         <Fill>
            <CssParameter name="fill">#ff0000</CssParameter>
         </Fill>
      </Mark>
   </Graphic>
</PointSymbolizer>
```

Fig. 7.22: *Figure 5: Setting the point color (#ff0000 = 100% red)*

Point Opacity

The opacity of a point is specified with a CssParameter element and a `fill-opacity` attribute. The opacity is specified as a floating point number between **0** and **1**, with 0 being completely transparent, and 1 being completely opaque.

```
<PointSymbolizer>
   <Graphic>
      <Mark>
         <Fill>
            <CssParameter name="fill-opacity">0.5</CssParameter>
```

```
        </Fill>
      </Mark>
   </Graphic>
</PointSymbolizer>
```

Fig. 7.23: *Figure 6: Setting the point opacity (0.5 = 50% opaque)*

Point Icon

An icon different from the default can be specified with the `ExternalGraphic` element:

```
<PointSymbolizer>
   <Graphic>
      <ExternalGraphic>
         <OnlineResource xlink:type="simple"
            xlink:href="http://maps.google.com/mapfiles/kml/pal3/icon55.png"/>
         <Format>image/png</Format>
      </ExternalGraphic>
   </Graphic>
</PointSymbolizer>
```

Fig. 7.24: *Figure 7: A custom icon for points*

In Figure 7, the custom icon is specified as a remote URL. It is also possible to place the graphic in the GeoServer `styles` directory, and then specify the filename only:

```
<PointSymbolizer>
   <Graphic>
      <ExternalGraphic>
         <OnlineResource xlink:type="simple" xlink:href="icon55.png"/>
         <Format>image/png</Format>
      </ExternalGraphic>
   </Graphic>
</PointSymbolizer>
```

Figure 8: Specifying a local file for a graphic point

Lines

Styles for lines are specified via a `LineSymbolizer`. An empty `LineSymbolizer` element will result in a default KML style:

```
<LineSymbolizer>
</LineSymbolizer>
```

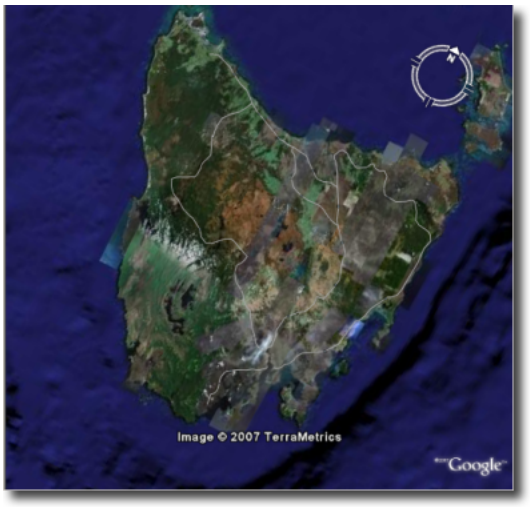


Fig. 7.25: *Figure 9: Default line*

The aspects of the resulting line which can be specified via a `LineSymbolizer` are *color*, *width*, and *opacity*.

Line Color

The color of a line is specified with a `CssParameter` element and a `stroke` attribute. The color is specified as a six digit hexadecimal code.

```
<LineSymbolizer>
   <Stroke>
      <CssParameter name="stroke">#ff0000</CssParameter>
   </Stroke>
</LineSymbolizer>
```

Line Width

The width of a line is specified with a `CssParameter` element and a `stroke-width` attribute. The width is specified as an integer (in pixels):

```
<LineSymbolizer>
   <Stroke>
      <CssParameter name="stroke-width">5</CssParameter>
```

Fig. 7.26: *Figure 10: Line rendered with color #ff0000 (100% red)*

```
   </Stroke>
</LineSymbolizer>
```

Fig. 7.27: *Figure 11: Line rendered with a width of five (5) pixels*

Line Opacity

The opacity of a line is specified with a `CssParameter` element and a `fill-opacity` attribute. The opacity is specified as a floating point number between **0** and **1**, with 0 being completely transparent, and 1 being completely opaque.

```
<LineSymbolizer>
   <Stroke>
      <CssParameter name="stroke-opacity">0.5</CssParameter>
   </Stroke>
</LineSymbolizer>
```

Fig. 7.28: *Figure 12: A line rendered with 50% opacity*

Polygons

Styles for polygons are specified via a `PolygonSymbolizer`. An empty `PolygonSymbolizer` element will result in a default KML style:

```
<PolygonSymbolizer>
</PolygonSymbolizer>
```

Polygons have more options for styling than points and lines, as polygons have both an inside ("fill") and an outline ("stroke"). The aspects of polygons that can be specified via a `PolygonSymbolizer` are *stroke color*, *stroke width*, *stroke opacity*, *fill color*, and *fill opacity*.

Polygon Stroke Color

The outline color of a polygon is specified with a `CssParameter` element and `stroke` attribute inside of a `Stroke` element. The color is specified as a 6 digit hexadecimal code:

```
<PolygonSymbolizer>
   <Stroke>
      <CssParameter name="stroke">#0000FF</CssParameter>
   </Stroke>
</PolygonSymbolizer>
```

Polygon Stroke Width

The outline width of a polygon is specified with a `CssParameter` element and `stroke-width` attribute inside of a `Stroke` element. The width is specified as an integer.

```
<PolygonSymbolizer>
   <Stroke>
      <CssParameter name="stroke-width">5</CssParameter>
   </Stroke>
</PolygonSymbolizer>
```

Fig. 7.29: *Figure 13: Outline of a polygon (#0000FF or 100% blue)*

Figure 14: Polygon with stroke width of five (5) pixels

Polygon Stroke Opacity

The stroke opacity of a polygon is specified with a `CssParameter` element and `stroke` attribute inside of a `Stroke` element. The opacity is specified as a floating point number between **0** and **1**, with 0 being completely transparent, and 1 being completely opaque.

```
<PolygonSymbolizer>
   <Stroke>
      <CssParameter name="stroke-opacity">0.5</CssParameter>
   </Stroke>
</PolygonSymbolizer>
```

Fig. 7.30: *Figure 15: Polygon stroke opacity of 0.5 (50% opaque)*

Polygon Fill Color

The fill color of a polygon is specified with a `CssParameter` element and `fill` attribute inside of a `Fill` element. The color is specified as a six digit hexadecimal code:

```
<PolygonSymbolizer>
   <Fill>
      <CssParameter name="fill">#0000FF</CssParameter>
   </Fill>
</PolygonSymbolizer>
```

Polygon Fill Opacity

The fill opacity of a polygon is specified with a `CssParameter` element and `fill-opacity` attribute inside of a `Fill` element. The opacity is specified as a floating point number between **0** and **1**, with 0 being completely transparent, and 1 being completely opaque.

Fig. 7.31: *Figure 16: Polygon fill color of #0000FF (100% blue)*

```
<PolygonSymbolizer>
   <Fill>
      <CssParameter name="fill-opacity">0.5</CssParameter>
   </Fill>
</PolygonSymbolizer>
```

Fig. 7.32: *Figure 17: Polygon fill opacity of 0.5 (50% opaque)*

Text Labels

There are two ways to specify a label for a feature in Google Earth. The first is with Freemarker templates (LINK?), and the second is with a `TextSymbolizer`. Templates take precedence over symbolizers.

Freemarker Templates

Specifying labels via a Freemarker template involves creating a special text file called `title.ftl` and placing it into the `workspaces/<ws name>/<datastore name>/<feature type name>` directory (inside the GeoServer data directory) for the dataset to be labeled. For example, to create a template to label the `states` dataset by state name one would create the file here: `<data_dir>/workspaces/topp/states_shapefile/states/title.ftl`. The content of the file would be:

```
${STATE_NAME.value}
```

Fig. 7.33: *Figure 18: Using a Freemarker template to display the value of STATE_NAME*

For more information on Placemark Templates, please see our full tutorial (LINK FORTHCOMING).

TextSymbolizer

In SLD labels are specified with the Label element of a `TextSymbolizer`. (Note the `ogc:` prefix on the `PropertyName` element.)

```
<TextSymbolizer>
   <Label>
      <ogc:PropertyName>STATE_NAME</ogc:PropertyName>
   </Label>
</TextSymbolizer>
```

The aspects of the resulting label which can be specified via a `TextSymbolizer` are *color* and *opacity*.

TextSymbolizer Color

The color of a label is specified with a `CssParameter` element and `fill` attribute inside of a `Fill` element. The color is specified as a six digit hexadecimal code:

```
<TextSymbolizer>
   <Label>
      <ogc:PropertyName>STATE_NAME</ogc:PropertyName>
   </Label>
```

Fig. 7.34: *Figure 19: Using a TextSymbolizer to display the value of STATE_NAME*

```
   <Fill>
      <CssParameter name="fill">#000000</CssParameter>
   </Fill>
</TextSymbolizer>
```

Fig. 7.35: *Figure 20: TextSymbolizer with black text color (#000000)*

TextSymbolizer Opacity

The opacity of a label is specified with a `CssParameter` element and `fill-opacity` attribute inside of a `Fill` element. The opacity is specified as a floating point number between **0** and **1**, with 0 being completely transparent, and 1 being completely opaque.

```
<TextSymbolizer>
   <Label>
      <ogc:PropertyName>STATE_NAME</ogc:PropertyName>
```

```
   </Label>
   <Fill>
      <CssParameter name="fill-opacity">0.5</CssParameter>
   </Fill>
</TextSymbolizer>
```

Fig. 7.36: *Figure 21: TextSymbolizer with opacity 0.5 (50% opaque)*

Descriptions

When working with KML, each feature is linked to a description, accessible when the feature is clicked on. By default, GeoServer creates a list of all the attributes and values for the particular feature.

It is possible to modify this default behavior. Much like with featureType titles, which are edited by creating a `title.ftl` template, a custom description can be used by creating template called `description.ftl` and placing it into the feature type directory (inside the GeoServer data directory) for the dataset. For instance, to create a template to provide a description for the states dataset, one would create the file: `<data_dir>/workspaces/topp/states_shapefile/states/description.ftl`. As an example, if the content of the description template is:

```
This is the state of ${STATE_NAME.value}.
```

The resultant description will look like this:

It is also possible to create one description template for all featureTypes in a given namespace. To do this, create a `description.ftl` file as above, and save it as `<data_dir>/templates/<workspace>/description.ftl`. Please note that if a description template is created for a specific featureType that also has an associated namespace description template, the featureType template (i.e. the most specific template) will take priority.

One can also create more complex descriptions using a combination of HTML and the attributes of the data. A full tutorial on how to use templates to create descriptions is available in our page on KML Placemark Templates. (LINK?)

SLD Generation from CSS SLD Structure Points Lines Polygons Text Labels Descriptions

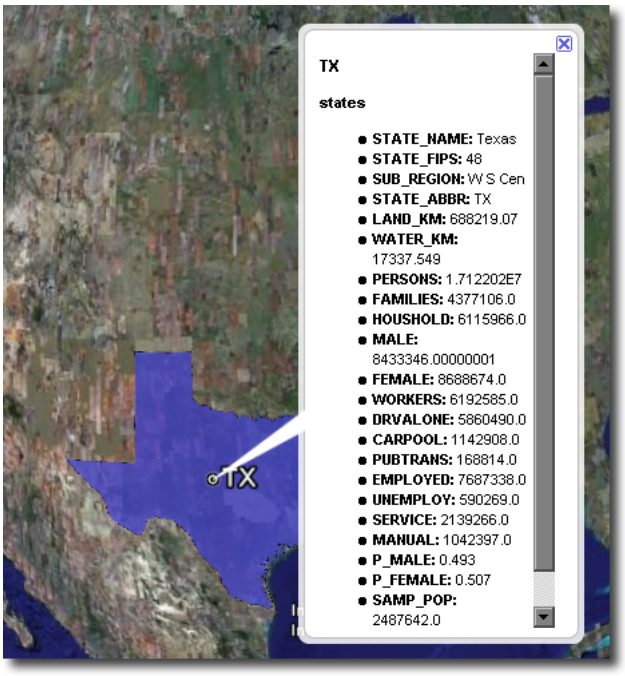


Fig. 7.37: *Figure 22: Default description for a feature*

Fig. 7.38: *Figure 23: A custom description*

Tutorials

KML Placemark Templates

Introduction

In KML a "Placemark" is used to mark a position on a map, often visualized with a yellow push pin. A placemark can have a "description" which allows one to attach information to it. Placemark descriptions are nothing more then an HTML snippet and can contain anything we want it to.

By default GeoServer produces placemark descriptions which are HTML tables describing all the attributes available for a particular feature in a dataset. In the following image we see the placemark description for the feature representing Idaho state:

Fig. 7.39: *The default placemark*

This is great, but what about if one wanted some other sort of information to be conveyed in the description. Or perhaps one does not want to show all the attributes of the dataset. The answer is Templates!!

A template is more or less a way to create some output.

Getting Started

First let us get set up. To complete the tutorial you will need the following:

- A GeoServer install
- A text editor

And thats it. For this tutorial we will assume that GeoServer is running the same configuration (data directory) that it does out of the box.

Hello World

Ok, time to get to creating our first template. We will start off an extremely simple template which, you guessed it, creates the placemark description "Hello World!". So lets go.

1. Using the text editor of your choice start a new file called `description.ftl`
2. Add the following content to the file:

```
Hello World!
```

3. Save the file in the `workspaces/topp/states_shapefile/states` directory of your "data directory". The data directory is the location of all the GeoServer configuration files. It is normally pointed to by the environment variable `GEOSERVER_DATA_DIR`.

4. Start GeoServer is it is not already running.

And thats it. We can now test out our template by adding the following network link in google earth:

```
http://localhost:8080/geoserver/wms/kml?layers=states
```

And voila. Your first template

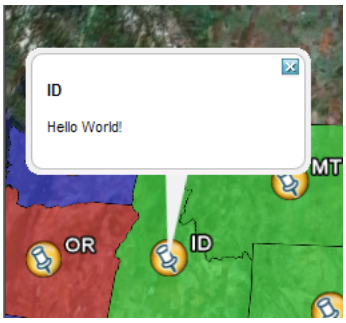


Fig. 7.40: *Hello World template.*

Refreshing Templates: One nice aspect of templates is that they are read upon every request. So one can simply edit the template in place and have it picked up by GeoServer as soon as the file is saved. So when after editing and saving a template simply "Refresh" the network link in Google Earth to have the new content picked up.

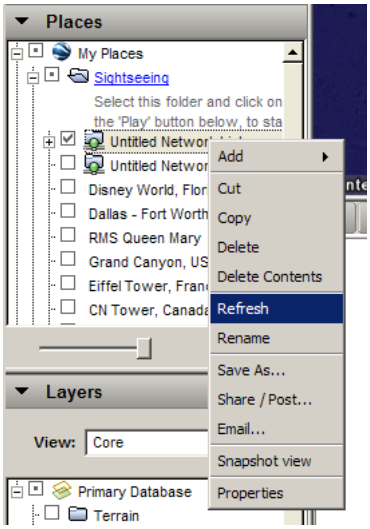


Fig. 7.41: *Refresh Template*

As stated before template descriptions are nothing more than html. Play around with `description.ftl` and add some of your own html. Some examples you may want to try:

1. A simple link to the homepage of your organization:

```
Provided by the <a href="http://topp.openplans.org">The Open Planning Project</a>.
```

Homepage of Topp

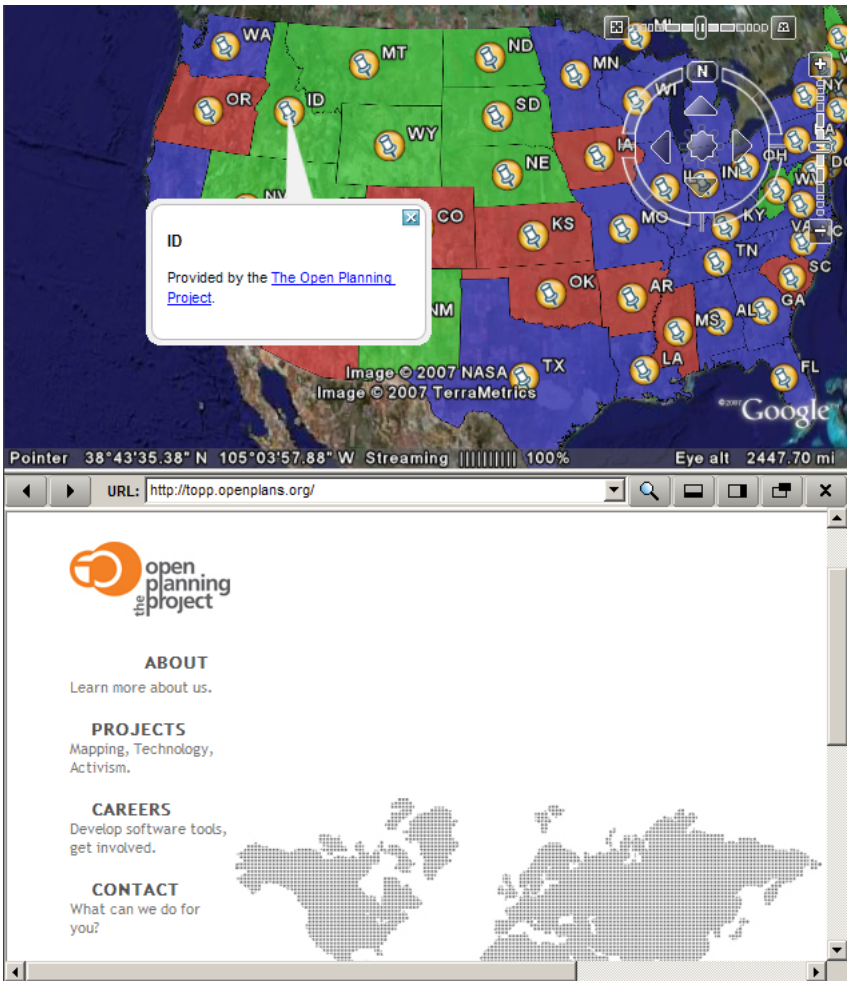


Fig. 7.42: *Homepage of Topp*

2. The logo of your organization:

```
<img src="http://topp.openplans.org/images/logo.jpg"/>
```

Logo of Topp

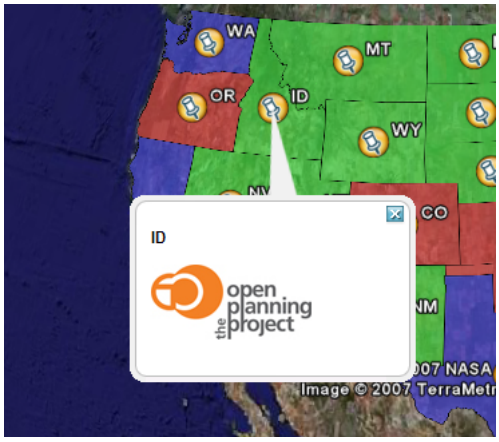


Fig. 7.43: *Logo of Topp*

The possibilities are endless. Now this is all great and everything but these examples are some what lacking in that the content is static. In the next section we will create more realistic template which actually access some the attributes of our data set.

Data Content

The real power of templates is the ability to easily access content, in the case of features this content is the attributes of features.In a KML placemark description template, there are a number of "template variables" available.

- The variable "fid", which corresponds to the id of the feature
- The variable "typeName", which corresponds to the name of the type of the feature
- A sequence of variables corresponding to feature attributes, each named the same name as the attribute

So with this knowledge in hand let us come up with some more examples:

Simple fid/typename access:

```
This is feature ${fid} of type ${typeName}.
```

This is a feature of 3.1 of type states.

Access to the values of two attributes named `STATE_NAME`, and `PERSONS`:

```
This is ${STATE_NAME.value} state which has a population of ${PERSONS.value}.
```

ID This is Idaho state which has a population of 1.006.749.

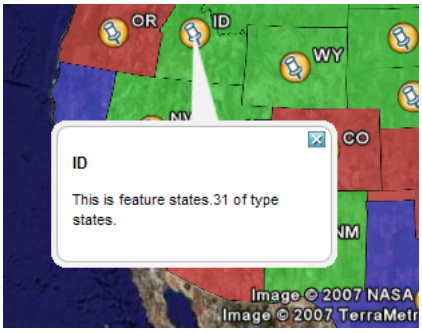


Fig. 7.44: *FID*

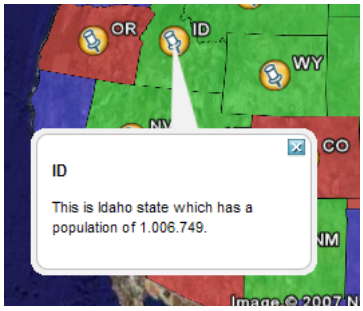


Fig. 7.45: *Attributes*

Attribute Variables

A feature attribute a "complex object" which is made up of three parts:

1. **A value**, given as a default string representation of the actual attribute value feasible to be used directly
2. **A rawValue**, being the actual value of the attribute, to allow for more specialized customization (for example, `${attribute.value?string("Enabled", "Disabled")}` for custom representations of boolean attributes, etc).
3. **A type**, each of which is accessible via `${<attribute_name>.name}`, `${<attribute_name>.value}`, `${<attribute_name>.rawValue}`, `${<attribute_name>.type}` respectively. The other variables: fid, and typeName and are "simple objects" which are available directly.

WMS Demo Example

We will base our final example off the "WMS Example" demo which ships with GeoServer. To check out the demo visit http://localhost:8080/geoserver/popup_map/index.html in your web browser.

You will notice that hovering the mouse over one of the points on the map displays an image specific to that point. Let us replicate this with a KML placemark description.

1. In the `featureTypes/DS_poi_poi` directory of the geoserver data directory create the following template:

```
<img src="http://localhost:8080/geoserver/popup_map/${THUMBNAIL.value}"/>
```

2. Add the following network link in Google Earth:

```
http://localhost:8080/geoserver/wms/kml?layers=tiger:poi
```

Poi.4

Fig. 7.46: *WMS Example*

Heights Templates

Introduction

Height Templates in KML allow you to use an attribute of your data as the 'height' of features in Google Earth.

Note: This tutorial assumes that GeoServer is running on http://localhost:8080.

Getting Started

For the purposes of this tutorial, you just need to have GeoServer with the release configuration, and Google Earth installed. Google Earth is available for free from <http://earth.google.com/ <http://earth.google.com/>'_.

Step One

By default GeoServer renders all features with 0 height, so they appear to lay flat on the world's surface in Google Earth.

To view the `topp:states` layer (packaged with all releases of GeoServer) in Google Earth, the easiest way is to use a network link. In Google Earth, under *Places*, right-click on *Temporary Places*, and go to *Add* → *Network Link*. In the dialog box, fill in `topp:states` as the *Name*, and the following URL as the *Link*:

```
http://localhost:8080/geoserver/wms/reflect?layers=topp:states&format=application/vnd.
↪google-earth.kml+xml
```

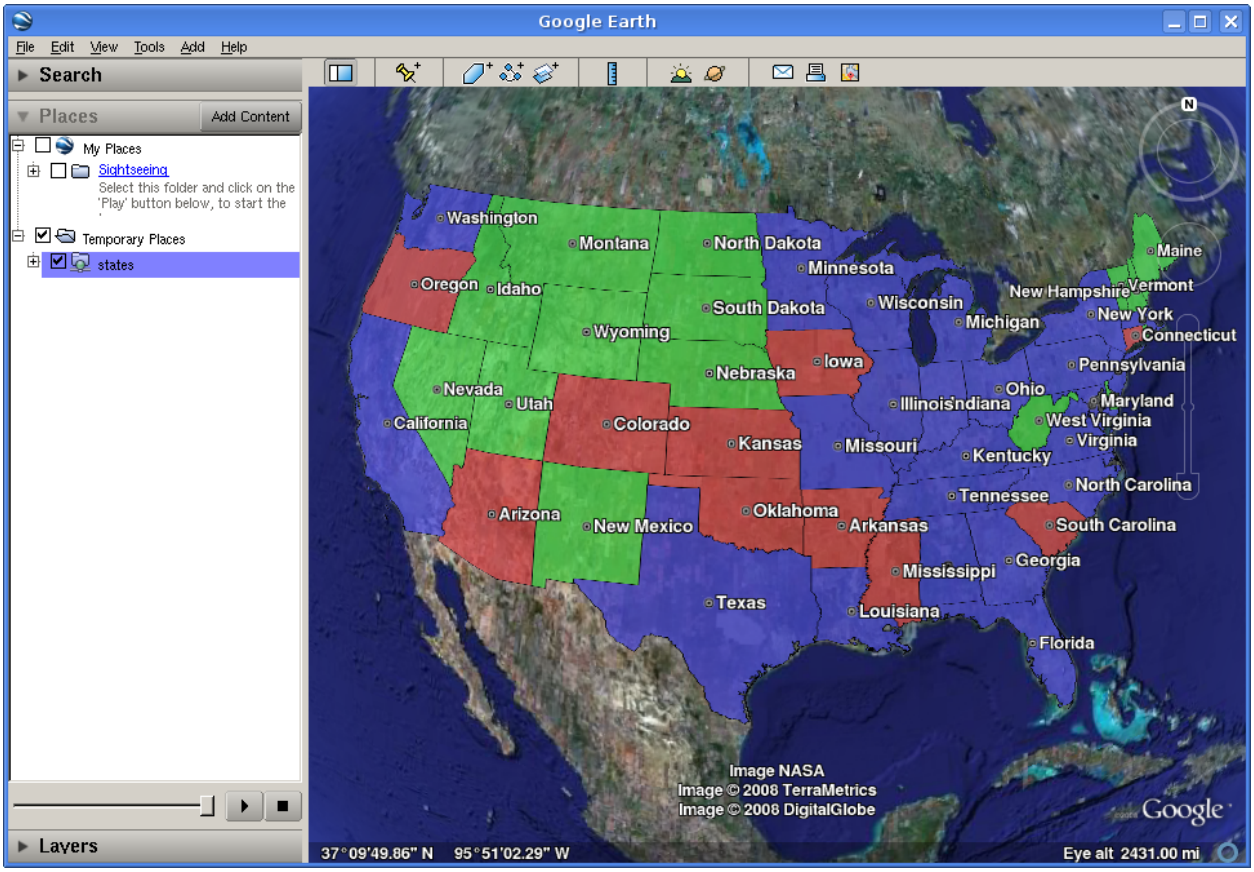


Fig. 7.47: *topp:states in Google Earth*

Step Two

An interesting value to use for the height would be the population of each state (so that more populated states appear taller on the map). We can do this by creating a file called `height.ftl` in the GeoServer data directory under `workspaces/topp/states_shapefile/states`. To set the population value, we enter the following text inside this new file:

```
${PERSONS.value}
```

This uses the value of the `PERSONS` attribute as the height for each feature. To admire our handiwork, we can refresh our view by right-clicking on our temporary place (called `topp:states`) and selecting *Refresh*:

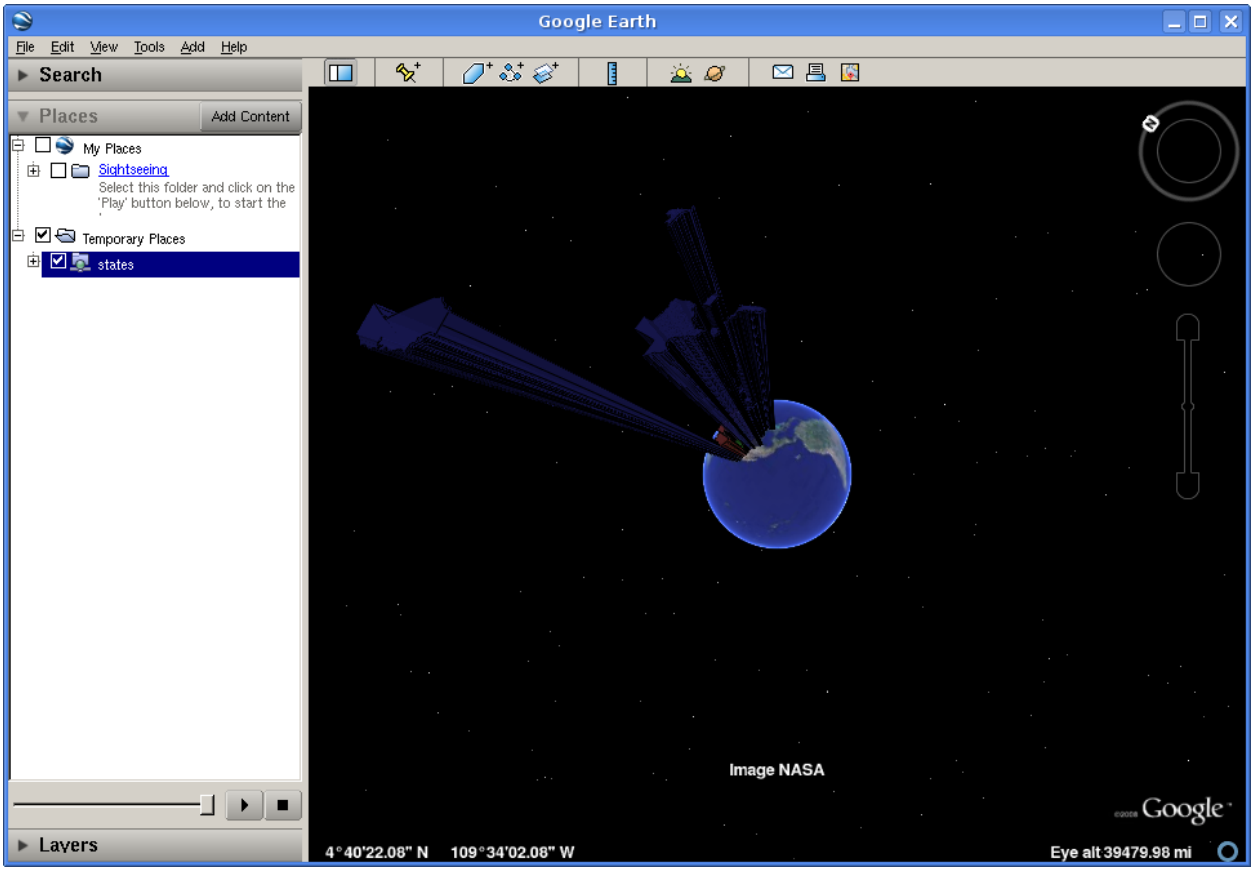


Fig. 7.48: *Height by Population*

Step Three

Looking at our population map, we see that California dwarfs the rest of the nation, and in general all of the states are too tall for us to see the heights from a convenient angle. In order to scale things down to a more manageable size, we can divide all height values by 100. Just change the template we wrote earlier to read:

```
${PERSONS.value / 100}
```

Refreshing our view once again, we see that our height field has disappeared. Looking at the GeoServer log (in the data directory under logs/geoserver.log) we see something like:

```
Caused by: freemarker.core.NonNumericalException: Error on line 1, column 3 in height.
↪ftl
Expression PERSONS.value is not numerical
```

However, we know that the `PERSONS` field is numeric, even if it is declared in the shapefile as a string value. To force a conversion, we can append `?number`, like so:

```
${PERSONS.value?number / 100}
```

One final *Refresh* brings us to a nicely sized map of the US:

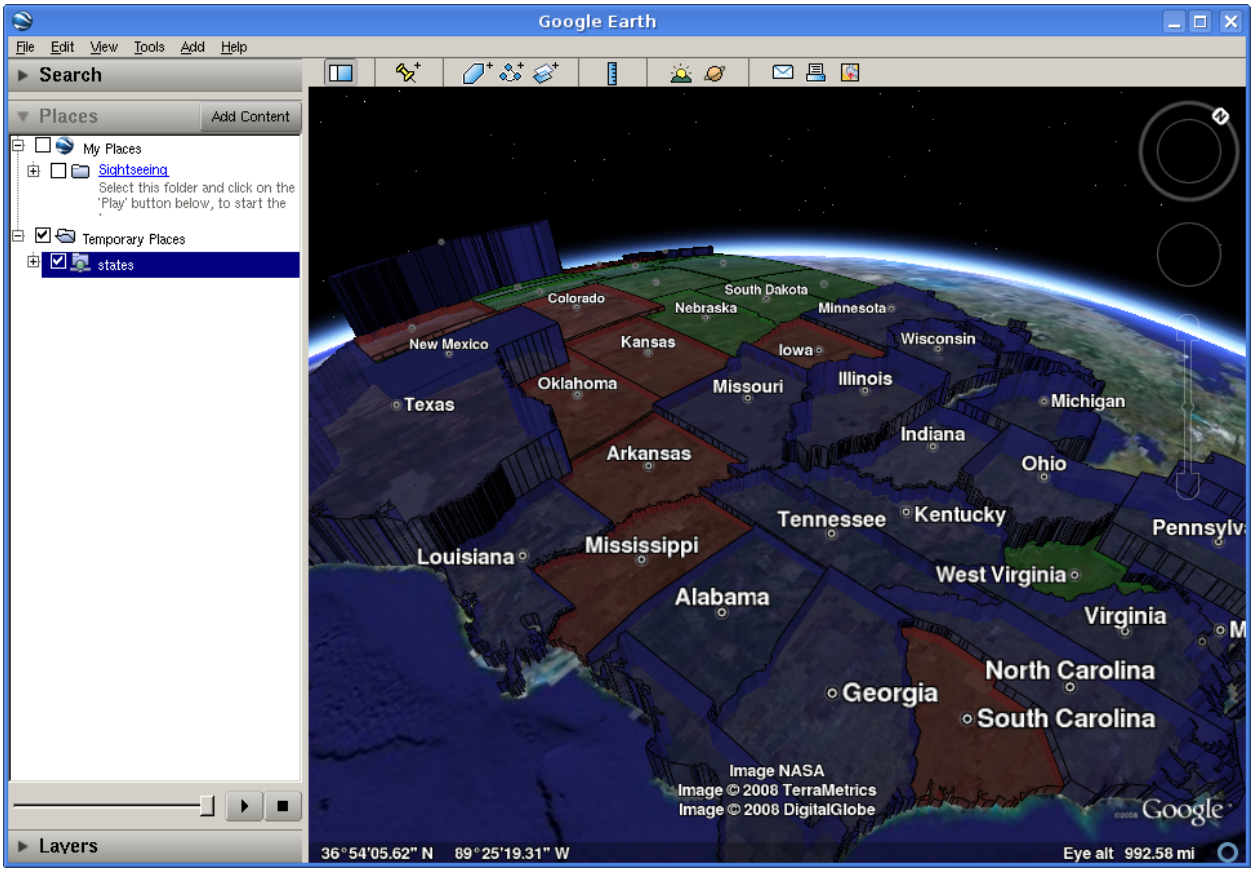


Fig. 7.49: *Scaled Height*

Step Four

There are still a couple of tweaks we can make. The default is to create a 'solid' look for features with height, but Google Earth can also create floating polygons that are disconnected from the ground. To turn off the 'connect to ground' functionality, add a format option called 'extrude' whose value is 'false'. That is, change the *Link* in the Network Link to be:

```
http://localhost:8080/geoserver/wms/reflect?layers=topp:states&format=application/vnd.
↪google-earth.kml%2Bxml&format_options=extrude:false
```

We also have a few options for how Google Earth interprets the height field. By default, the height is interpreted as relative to the ground, but we can also set the heights relative to sea level, or to be ignored (useful for reverting to the 'flat' look without erasing your template). This is controlled with a format option named altitudeMode, whose values are summarized below.

altitudeMode	Purpose
altitudeMode	Interpret height as relative to ground level
absolute	Interpret height as relative to sea level
clampToGround	Ignore height entirely

Time

Warning: The screenshots on this tutorial have not yet been updated for the 2.0.x user interface. But most all the rest of the information should be valid, and the user interface is roughly the same, but a bit more easy to use.

Getting Started

For this tutorial we will using a Shapefile which contains information about the number of Internet users in the countries of Western Europe for a rang of years.

1. Download and unzip `inet_weu.zip`
2. Configure GeoServer to serve the Shapefile `inet_weu.zip`. (A tutorial is available *Publishing a shapefile*.)
3. Add the SLD "`inet_weu.sld` to GeoServer. (A tutorial is available for *Styling*)
4. Set the style of the feature type added in step 2 to the style added in step 3

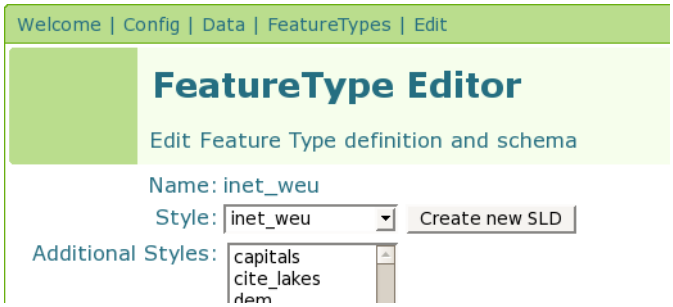


Fig. 7.50: *Style*

Checking the Setup

If all is configured properly you should be able to navigate to http://localhost:8080/geoserver/wms/kml?layers=topp:inet_weu&format=openlayers&bbox=-33.780,26.266,21.005,56.427 and see the following map:

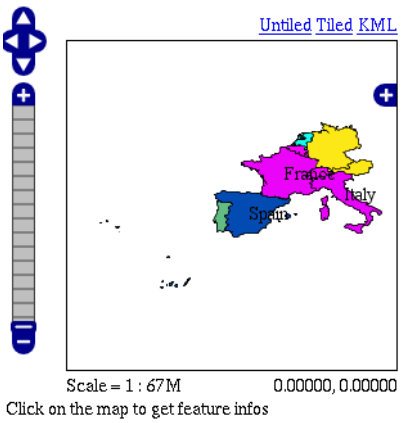


Fig. 7.51: *Setup*

Creating the Template

Next we will create a template which allows us to specify the temporal aspects of the dataset. The schema of our dataset looks like:

INET_P100n	Number of internet users per 100 people
NAME	Name of country
RPT_YEAR	Year
Geometry	Polygon representing the country

The temporal attribute is `RPT_YEAR` and is the one that matters to us. Ok, time to create the template.

1. In your text editor of choice, create a new text file called `time.ftl`.
2. Add the following text:

```
${RPT_YEAR.value?date('yyyy')}
```

3. Save the file to the `<GEOSERVER_DATA_DIR>/workspaces/topp/inet_weu_shapefile/inet_weu` directory. Where `<GEOSERVER_DATA_DIR>` is the location of the "data directory" of your GeoServer installation. Usually pointed to via the `GEOSERVER_DATA_DIR` environment variable.

See the ref:*references* section for more information about specifying a date format.

Trying it Out

Ok time to try it out.

1. Navigate to http://localhost:8080/geoserver/wms/kml?layers=inet_weu&legend=true. This should cause Google Earth to open.
2. In Google Earth, adjust the time bar so that it captures a time interval that is approximately 1 year wide
3. Slide the time bar forward in time and notice how the polygon colors change

Fig. 7.52: *Google Earth*

Fig. 7.53: *Google Earth Time Bar*

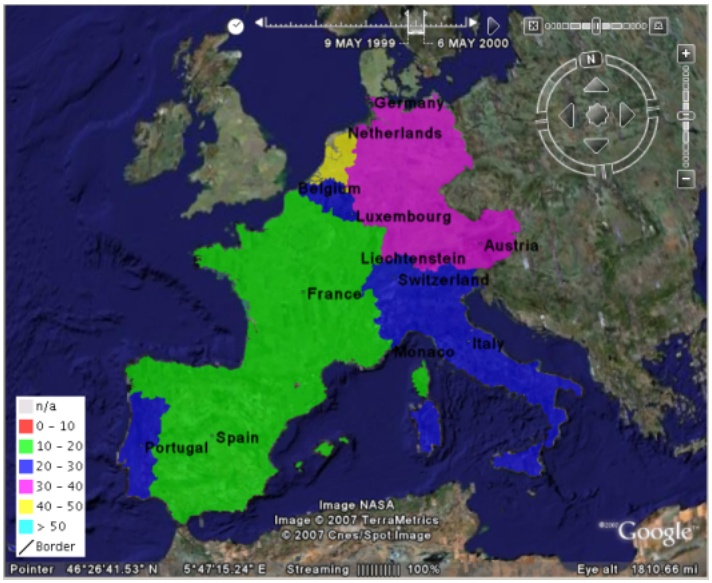


Fig. 7.54: *Sliding the Time Bar*

References

Specifying a Date Format

When setting up a time template for your own dataset the most important issue is the format of your temporal data. It may or may not be in a format in which GeoServer can read directly. You can check if the date/time format can be used directly by GeoServer by using the following time template. This is an example time template file (time.ftl) file without explicit formatting.

```
${DATETIME_ATTRIBUTE_NAME.value}
```

While GeoServer will try its best to parse the data there are cases in which your data is in a format which it cannot parse. When this occurs it is necessary to explicitly specify the format. Luckily Freemarker provides us with functionality to do just this.

Consider the date time `12:30 on January 01, 2007` specified in the following format: `01?01%2007&12$30!00`. When creating the template we need to explicitly tell Freemarker the format the date time is in with the datetime function. This is an example time template file (time.ftl) file with explicit formatting:

```
${DATETIME_ATTRIBUTE_NAME.value?datetime("M?d%y&H:m:s")}
```

The process is similar for dates (no time). The date `01?01%2007` would be specified in a template with explicit formatting:

```
${DATETIME_ATTRIBUTE_NAME.value?date("M?d%y")}
```

So when must you specify the date format in this manner? The following table illustrates the *date* formats that GeoServer can understand. Note that the '-' character can be one of any of the following characters: '/' (forward slash), ' ' (space), '.' (period), ',' (comma)

Date Format	Example
yyyy-MM-dd	2007-06-20
yyyy-MMM-dd	2007-Jun-20
MM-dd-yyyy	06-20-2007
MMM-dd-yyyy	Jun-20-2007
dd-MM-yyyy	20-06-2007
dd-MMM-yyyy	20-Jun-2007

The set of *date time* formats which GeoServer can be understand is formed by appending the timestamp formats `hh:mm` and `hh:mm:ss` to the entries in the above table:

DateTime Format	Example
yyyy-MM-dd hh:mm	2007-06-20 12:30
yyyy-MMM-dd hh:mm	2007-Jun-20 12:30
yyyy-MM-dd hh:mm:ss	2007-06-20 12:30:00
yyyy-MMM-dd hh:mm:ss	2007-Jun-20 12:30:00

Warning: Setting the Timezone

Be aware that the KML output for *date time* formats will reflect the timezone of the java virtual machine, which can be set using the user.timezone parameter in the startup script. For example, the following

command starts GeoServer using the Coordinated Universal Time (UTC) timezone.

```
exec "$_RUNJAVA" -DGEOSERVER_DATA_DIR="$GEOSERVER_DATA_DIR" -Djava.
    awt.headless=true -DSTOP.PORT=8079  -Duser.timezone=UTC  -DSTOP.
    KEY=geoserver -jar start.jar
```

If the timezone is not set, it will default to the timezone of the operating system.

Specifying a Date Range

In the above example a single time stamp is output for the dataset. GeoServer also supports specifying date ranges via a template. The syntax for ranges is:

Where begin is the first date in the range, end is the last date in the range, and `||` is the delimiter between the two. As an example:

Would the date range starting at `January 1, 2007` and ending `June 1, 2007`. Date ranges can also be open ended:

The first date specifies a date range where the beginning is open-ended. The second specifies a date range where the end is open-ended.

Super-Overlays and GeoWebCache

Overview

This tutorial explains how to use GeoWebCache (GWC) to enhance the performance of super-overlays in Google Earth. For more information please see the page on *KML Super-Overlays*

Conveniently GeoWebCache can generate super-overlays automatically. With the standalone GeoWebCache it takes minimal amount of configuration. Please see the GeoWebCache documentation for more information on the standalone version of GeoWebCache.

We are going to use the plug in version of GeoWebCache where there is no configuration need. For this tutorial we are also using the topp:states layer. Using the GeoWebCache plug in with super-overlays

To access GWC from GeoServer go to http://localhost:8080/geoserver/gwc/demo/. This should return a layer list of similar to below.

To use a super-overlay in GeoWebCache select the KML (vector) option display for each layer. Lets select topp:states.The url would be http://localhost:8080/geoserver/gwc/service/kml/topp:states.kml.kmz After doing so you will be presented with a open option dialog, choose Google Earth.

When Google Earth finishes loading you should be viewing a the topp:states layers.

Features

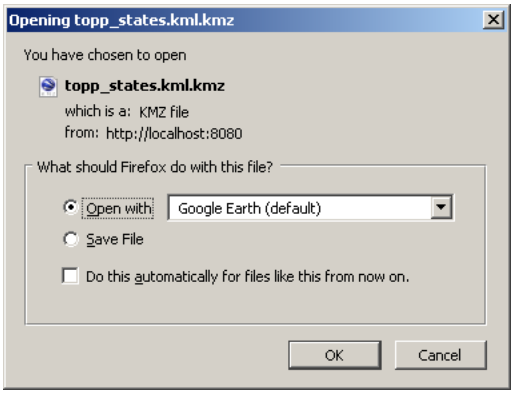
This section delves into greater detail about the various functionality and options possible with KML output and Google Earth.

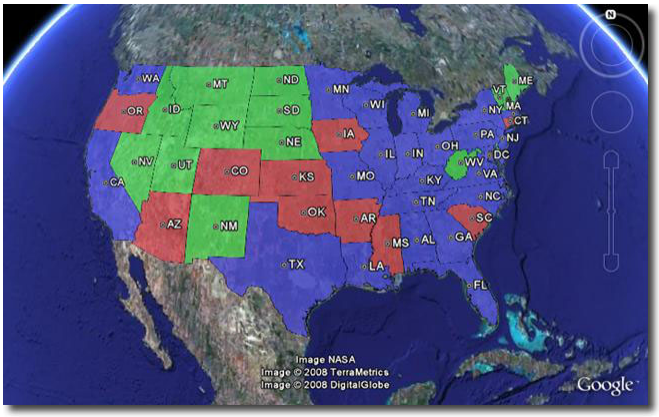
KML Reflector

Standard WMS requests can be quite long and cumbersome. The following is an example of a request for KML output from GeoServer:

GeoWebCache

Layer name:	Enabled:	Grids Sets:	
nurc:Arc_Sample Seed this layer	true	EPSG:4326 EPSG:900913	OpenLayers: [png, jpeg] KML: [png, jpeg] OpenLayers: [png, jpeg]
nurc:Img_Sample Seed this layer	true	EPSG:4326 EPSG:900913	OpenLayers: [png, jpeg] KML: [png, jpeg] OpenLayers: [png, jpeg]
nurc:Pk50095 Seed this layer	false	EPSG:4326 EPSG:900913	OpenLayers: [png, jpeg] KML: [png, jpeg] OpenLayers: [png, jpeg]
nurc:mosaic Seed this layer	true	EPSG:4326 EPSG:900913	OpenLayers: [jpeg, png] KML: [jpeg, png] OpenLayers: [jpeg, png]
sf:archsites Seed this layer	true	EPSG:4326 EPSG:900913	OpenLayers: [png, jpeg] KML: [png, jpeg] OpenLayers: [png, jpeg]
sf:bugsites Seed this layer	true	EPSG:4326 EPSG:900913	OpenLayers: [png, jpeg] KML: [png, jpeg] OpenLayers: [png, jpeg]
sf:restricted	true		





```
http://localhost:8080/geoserver/ows?service=WMS&request=GetMap&version=1.1.1&
↪format=application/vnd.google-earth.kml+XML&width=1024&height=1024&srs=EPSG:4326&
↪layers=topp:states&styles=population&bbox=-180,-90,180,90
```

GeoServer includes an alternate way of requesting KML, and that is to use the **KML reflector**. The KML reflector is a simpler URL-encoded request that uses sensible defaults for many of the parameters in a standard WMS request. Using the KML reflector one can shorten the above request to:

```
http://localhost:8080/geoserver/wms/kml?layers=topp:states
```

Using the KML reflector

The only mandatory parameter is the `layers` parameter. The syntax is as follows:

```
http://GEOSERVER_URL/wms/kml?layers=<layer>
```

where `GEOSERVER_URL` is the URL of your GeoServer instance, and `<layer>` is the name of the featuretype to be served.

The following table lists the default assumptions:

Key	Value
request	GetMap
service	wms
version	1.1.1
srs	EPSG:4326
format	applcation/vnd.google-earth.kmz+xml
width	2048
height	2048
bbox	<layer bounds>
kmattr	true
kmplacemark	false
kmscore	40
styles	[default style for the featuretype]

Any of these defaults can be changed when specifying the request. For instance, to specify a particular style, one can append `styles=population` to the request:

```
http://localhost:8080/geoserver/wms/kml?layers=topp:states&styles=population
```

To specify a different bounding box, append the parameter to the request:

```
http://localhost:8080/geoserver/wms/kml?layers=topp:states&bbox=-124.73,24.96,-66.97,
↪49.37
```

Reflector modes

The KML reflector can operate in one of three modes: **refresh**, **superoverlay**, and **download**.

The mode is set by appending the following parameter to the URL:

```
mode=<mode>
```

where <mode> is one of the three reflector modes. The details for each mode are as follows:

Mode	Description
refresh	(*default for all versions except 1.7.1 through 1.7.5*) Returns dynamic KML that can be refreshed/updated by the Google Earth client. Data is refreshed and new data/imagery is downloaded when zooming/panning stops. This mode can return either vector or raster (placemark or overlay) The decision to return either vector or raster data is determined by the value of kmscore. Please see the section on *KML Scoring* for more information.
superoverlay	(*default for versions 1.7.1 through 1.7.5*) Returns KML as a super-overlay. A super-overlay is a form of KML in which data is broken up into regions. Please see the section on *KML Super-Overlays* for more information.
download	Returns KML which contains the entire data set. In the case of a vector layer, this will include a series of KML placemarks. With raster layers, this will include a single KML ground overlay. This is the only mode that doesn't dynamically request new data from the server, and thus is self-contained KML.

More about the "superoverlay" mode

When requesting KML using the superoverlay mode, there are four additional submodes available regarding how and when data is requested. These options are set by appending the following parameter to the KML reflector request:

```
superoverlay_mode=<submode>
```

where <submode> is one of the following options:

Submode	Description
auto	(*default*) Always returns vector features if the original data is in vector form, and returns raster imagery if the original data is in raster form. This can sometimes be less than optimal if the geometry of the features are very complicated, which can slow down Google Earth.
raster	Always returns raster imagery, regardless of the original data. This is almost always faster, but all vector information is lost in this view.
overview	Displays either vector or raster data depending on the view. At higher zoom levels, raster imagery will be displayed, and at lower zoom levels, vector features will be displayed. The determination for when to switch between vector and raster is made by the regionation parameters set on the server. See the section on *KML Regionation* for more information.
hybrid	Displays both raster and vector data at all times.

Toggling Placemarks

Vector Placemarks

When GeoServer generates KML for a vector dataset, it attaches information from the data to each feature that is created. When clicking on a vector feature, a pop-up window is displayed. This is called a **placemark**.

By default this is a simple list which displays attribute data, although this information can be customized using Freemarker templates.

If you would like this information not to be shown when a feature is clicked (either for security reasons, or simply to have a cleaner user interface), it is possible to disable this functionality. To do so, use the `kmattr` parameter in a KML request to turn off attribution.

The syntax for `kmattr` is as follows:

```
format_options=kmattr:[true|false]
```

Note that `kmattr` is a "format option", so the syntax is slightly different from the usual key-value pair. For example:

```
http://localhost:8080/geoserver/wms/kml?layers=topp:states&format_options=kmattr:false
```

Raster Placemarks

Unlike vector features, where the placemark is enabled by default, placemarks are disabled by default with raster images of features. To enable this feature, you can use the `kmplacemark` format option in your KML request. The syntax is similar to the `kmattr` format option specified above:

```
format_options=kmplacemark:[true|false]
```

For example, using the KML reflector, the syntax would be:

```
http://localhost:8080/geoserver/wms/kml?layers=topp:states&format_
↪options=kmplacemark:true
```

Customizing Placemarks

KML output can leverage some powerful visualization abilities in Google Earth. **Titles** can be displayed on top of the features. **Descriptions** (custom HTML shown when clicking on a feature) can be added to customize the views of the attribute data. In addition, using Google Earth's time slider, **time**-based animations can be created. Finally, **height** of features can be set, as opposed to the default ground overlay. All of these can be accomplished by creating Freemarker templates. Freemarker templates are text files (with limited HTML code), saved in the *GeoServer data directory*, that utilize variables that link to specific attributes in the data.

Titles

Specifying labels via a template involves creating a special text file called `title.ftl` and placing it into the featuretypes directory inside the *GeoServer data directory* for the dataset to be labeled. For instance, to create a template to label the `states` layer by state name, one would create the file: `<data_dir>/workspaces/topp/states_shapefile/states/title.ftl`. The content of the file would be:

```
${STATE_NAME.value}
```

Descriptions

When working with KML, each feature is linked to a description, accessible when the feature is clicked on. By default, GeoServer creates a list of all the attributes and values for the particular feature.

It is possible to modify this default behavior. Much like with featuretype titles, which are edited by creating a `title.ftl` template, specifying descriptions via a template involves creating a special text file called `description.ftl` and placing it into the featuretypes directory inside the *GeoServer data directory* for the dataset to be labeled. For instance, a sample description template would be saved here: `<data_dir>/workspaces/topp/states_shapefile/states/description.ftl`. The content of the file could be:

```
This is the state of ${STATE_NAME.value}.
```

The resulting description will look like this:

Warning: Add SS: A custom description

It is also possible to create one description template for all layers in a given namespace. To do this, create a `description.ftl` file as above, and save it here:

```
<data_dir>/templates/<namespace>/description.ftl.
```

Please note that if a description template is created for a specific layer that also has an associated namespace description template, the layer template (i.e. the most specific template) will take priority.

KML Placemark Placement

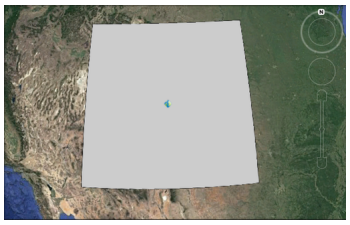
The placemark "placement" (also referred to as the "centroid") refers to the location of the placemark icon with respect to the feature geometry itself. Historically this placement has been chosen to be simply the centroid of the feature geometry. This section describes options for controlling placement were introduced.

Clipping

The `KMCENTROID_CLIP` option determines whether the feature geometry should be clipped to the viewport before the centroid is calculated. This will ensure that the placemark icon always falls within the viewport, even in cases part of a geometry falls outside of it.

The option is set to either `true` or `false`. The default value is"false".

As an example consider the following square polygon with it's placemark icon. When the polygon is entirely in the viewport the placement is good.



When the polygon moves out of the viewport the icon is lost as in the following figure:

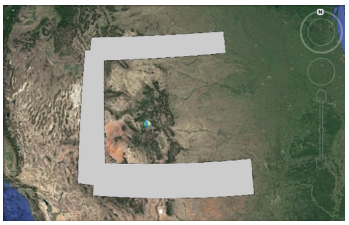
When `KMCENTROID_CLIP` set to `true` only the part of the geometry intersecting the viewport is considered.

Sampling for internal point

The `KMCENTROID_CONTAIN` option determines whether point chosed for the centroid must fall within the feature geometry. For irregularily shaped geometries (like a "C" shaped polygon) the default centroid calculation will fall outside of the geometry. The option is set to either `true` or `false`. The default value is `false`.

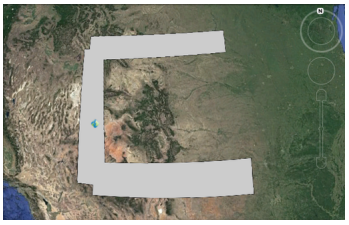
In order to find a contained point of a polygon a sampling tequnique is used where a number of points are chosen until one is found that falls within the polygon. The `KMCENTROID_SAMPLE` option determines how many samples to attempt. The value is an integer with a default value is 5. Note that this option only applies when `KMCENTROID_CONTAIN` is set to `true`.

As an example consider the following "c" shaped polygon with it's placemark icon. By default the icon falls outside of the polygon.



When `KMCENTROID_CONTAIN` set to `true` a point within the polygon is chosen.

Note: The sampling tequnique may not always be able to find a suitable point. You can try to increase the number of samples but it is still not a guarantee. Care also must be taken when increasing the sample count since it adds overhead to the overall KML rendering process.



KML Height and Time

Height

GeoServer by default creates two dimensional overlays in Google Earth. However, GeoServer can output features with height information (also called "KML extrudes") if desired. This can have the effect of having features "float" above the ground, or create bar graph style structures in the shape of the features. The height of features can be linked to an attribute of the data.

Setting the height of features is determined by using a KML Freemarker template. Create a file called `height.ftl`, and save it in the same directory as the featuretype in your *GeoServer data directory*. For example, to create a height template for the `states` layer, the file should be saved in `<data_dir>/workspaces/topp/states_shapefile/states/height.ftl`.

To set the height based on an attribute, the syntax is:

```
${ATTRIBUTE.value}
```

Replace the word `ATTRIBUTE` with the name of the height attribute in your data set. For a complete tutorial on working with the height templates see *Heights Templates*.

Time

Google Earth also contains a "time slider", which can allow animations of data, and show changes over time. As with height, time can be linked to an attribute of the data, as long as the data set has a date/time attribute. Linking this date/time attribute to the time slider in Google Earth is accomplished by creating a Freemarker template. Create a file called `time.ftl`, and save it in the same directory that contains your data's `info.xml`.

To set the time based on an attribute the syntax is:

```
${DATETIME_ATTRIBUTE.value}
```

Replace the word `DATETIME_ATTRIBUTE` with the name of the date/time attribute. When creating KML, GeoServer will automatically link the data to the time element in Google Earth. If set successfully, the time slider will automatically appear.

For a full tutorial on using GeoServer with Google Earth's time slider see *Time*

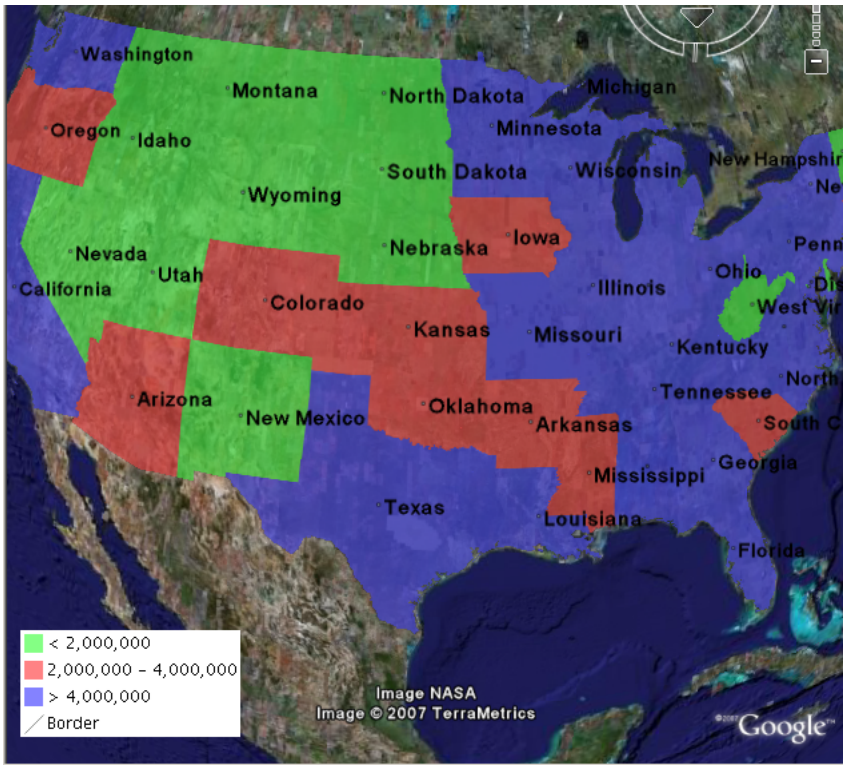
KML Legends

WMS includes a `GetLegendGraphic` operation which allows a WMS client to obtain a legend graphic from the server for a particular layer. Combining the legend with KML overlays allows the legend to be viewed inside Google Earth.

To get GeoServer to include a legend with the KML output, append `legend=true` to the KML reflector request. For example:

```
http://localhost:8080/geoserver/wms/kml?layers=topp:states&legend=true
```

The resulting Google Earth output looks like this:



Filters

Though not specific to Google Earth, GeoServer has the ability to filter data returned from the *Web Map Service (WMS)*. The KML Reflector will pass through any WMS `filter` or `cql_filter` parameter to GeoServer to constrain the response.

Note: Filters are basically a translation of a SQL "WHERE" statement into web form. Though limited to a single table, this allows users to do logical filters like "AND" and "OR" to make very complex queries, leveraging numerical and string comparisons, geometric operations ("bbox", "touches", "intersects", "disjoint"), "LIKE" statements, nulls, and more.

Filter

There simplest filter is very easy to include. It is called the `featureid` filter, and it lets you filter to a single feature by its ID. The syntax is:

```
featureid=<feature>
```

where <feature> is the feature and its ID. An example would be:

```
http://localhost:8080/geoserver/wms/kml?layers=topp:states&featureid=states.5
```

This request will output only the state of Maryland. The feature IDs of your data are most easily found by doing WFS or KML requests and examining the resulting output.

CQL Filter

Using filters in a URL can be very unwieldy, as one needs to include URL-encoded XML:

```
http:/localhost:8080/geoserver/wms/kml?layers=topp:states&FILTER=%3CFilter%3E
↪%3CPropertyIsBetween%3E%3CPropertyName%3Etopp:LAND_KM%3C/PropertyName%3E
↪%3CLowerBoundary%3E%3CLiteral%3E100000%3C/Literal%3E%3C/LowerBoundary%3E
↪%3CUpperBoundary%3E%3CLiteral%3E150000%3C/Literal%3E%3C/UpperBoundary%3E%3C/
↪PropertyIsBetween%3E%3C/Filter%3E
```

Instead, one can use Common Query Language (CQL), which allows one to specify the same statement more succinctly:

```
http://localhost:8080/geoserver/wms/kml?layers=topp:states&CQL_FILTER=LAND_
↪KM+BETWEEN+100000+AND+150000
```

This query will return all the states in the US with areas between 100,000 and 150,000 km^2.

KML Super-Overlays

Super-overlays are a form of KML in which data is broken up into regions. This allows Google Earth to refresh/request only particular regions of the map when the view area changes. Super-overlays are used to efficiently publish large sets of data. (Please see Google's page on super-overlays for more information.)

GeoServer supports two types of super-overlays: **raster** and **vector**. With raster super-overlays, GeoServer intelligently produces imagery appropriate to the current zoom level and dynamically outputs new imagery when the zoom level changes. With vector super-overlays, feature data is requested for only the visible features and new features are dynamically loaded as necessary. Raster super-overlays require less resources on the client, but vector super-overlays have a higher output quality.

When using the *KML Reflector*, super-overlays are enabled by default, whether the data in question is raster or vector. For more information on the various options for KML super-overlay output, please see the page on the *KML Reflector*.

Raster Super-Overlays

Consider this image, which is generated from GeoServer. When zoomed out, the data is at a small size.

When zooming in, the image grows larger, but since the image is at low resolution (orignially designed to be viewed small), the quality degrades.

However, in a super-overlay, the KML document requests a new image from GeoServer of a higher resolution for that zoom level. As the new image is downloaded, the old image is replaced by the new image.

Raster Super-Overlays and GeoWebCache

GeoServer implements super-overlays in a way that is compatible with the WMS Tiling Client Recommendation. Super-overlays are generated such that the tiles of the super-overlay are the same tiles that a WMS tiling client would request. One can therefore use existing tile caching mechanisms and reap a potentially large performance benefit.

The easiest way to tile cache a raster super overlay is to use GeoWebCache which is built into GeoServer:

```
http://GEOSERVER_URL/gwc/service/kml/<layername>.<imageformat>.kmz
```

where `GEOSERVER_URL` is the URL of your GeoServer instance.

Vector Super-Overlays

GeoServer can include the feature information directly in the KML document. This has lots of benefits. It allows the user to select (click on) features to see descriptions, toggle the display of individual features, as well as have better rendering, regardless of zoom level. For large datasets, however, the feature information

can take a long time to download and use a lot of client-side resources. Vector super-overlays allow the client to only download part of a dataset, and request more features as necessary.

Vector super-overlays can use the process of *KML Regionation* to organize features into a hierarchy. The regionation process can operate in a variety of modes. Most of the modes require a "regionation attribute" which is used to determine which features should be visible at a particular zoom level. Please see the *KML Regionation* page for more details.

Vector Super-Overlays and GeoWebCache

As with raster super-overlays, it is possible to cache vector super-overlays using GeoWebCache. Below is the syntax for generating a vector super-overlay KML document via GeoWebCache:

```
http://GEOSERVER_URL/gwc/service/kml/<layername>.kml.kmz
```

where `GEOSERVER_URL` is the URL of your GeoServer instance.

Unlike generating a super-overlay with the standard *KML Reflector*, it is not possible to specify the regionation properties as part of the URL. These parameters must be set in the *Layers* configuration which can be navigated to by clicking on 'Layers' in the left hand sidebar and then selecting your vector layer.

KML Regionation

Displaying vector features on Google Earth is a very powerful way of creating nicely-styled maps. However, it is not always optimal to display all features at all times. Displaying too many features can create an unsightly map, and can adversely affect Google Earth's performance. To combat this, GeoServer's KML output includes the ability to limit features based on certain criteria. This process is known as **regionation**. Regionation is active by default when using the superoverlay KML reflector mode.

Regionation Attributes

The most important aspect of regionation is to decide how to determine which features show up more prominently than others. This can be done either **by geometry**, or **by attribute**. One should choose the option that best exemplifies the relative "importance" of the feature. When choosing to regionate by geometry, only the larger lines and polygons will be displayed at higher zoom levels, with smaller ones being displayed when zooming in. When regionating by an attribute, the higher value of this attribute will make those features show up at higher zoom levels. (Choosing an attribute with a non-numeric value will be ignored, and will instead default to regionation by geometry.)

Regionation Strategies

Regionation strategies sets how to determine which features should be shown at any given time or zoom level. There are five types of regionation strategies:

Strategy	Description
`best_guess`	(*default*) The actual strategy is determined by the type of data being operated on. If the data consists of points, the `random` strategy is used. If the data consists of lines or polygons, the `geometry` strategy is used.
`external-sorting`	Creates a temporary auxiliary database within GeoServer. It takes slightly extra time to build the index upon first request.
`native-sorting`	Uses the default sorting algorithm of the backend where the data is hosted. It is faster than external-sorting, but will only work with PostGIS datastores.
`geometry`	Externally sorts by length (if lines) or area (if polygons).
`random`	Uses the existing order of the data and does not sort.

In most cases, the **best_guess** strategy is sufficient.

Setting Regionation Parameters

Regionation strategies and attributes are featuretype-specific, and therefore are set in the *Layers* editing page of the *Web administration interface*. This can be navigated to by selecting 'Layers' on the left sidebar.

KML Scoring

Note: KML scoring only applies when using the super-overlay mode `refresh`. See *KML Super-Overlays* for more information.

GeoServer can return KML in one of two forms. The first is as a number of placemark elements (vectors). Each placemark corresponds to a feature in the underlying dataset. This form only applies to vector datasets.

The second form is as an overlay (image). In this form the rendering is done by the GeoServer WMS and only the resulting graphic is sent to Google Earth. This is the only form available for raster datasets, but can be applied to vector datasets as well.

There are advantages to and disadvantages to each output mode when rendering vector data. Placemarks look nicer, but there can be performance problems with Google Earth if the data set is large. Overlays put less of a strain on Google Earth, but aren't as nice looking.

The following shows the same dataset rendered in Placemark form on the top and Overlay form on the bottom.

KML scoring is the process of determing whether to render features as rasters or as vectors.

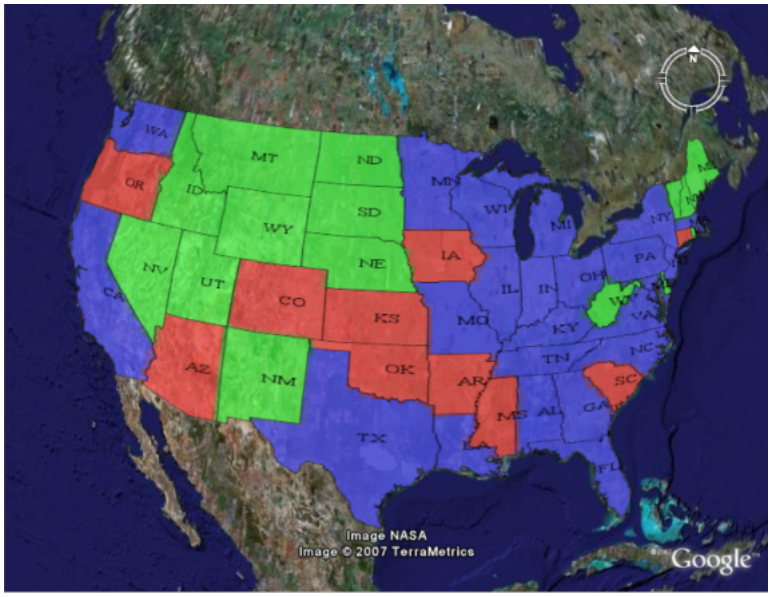
The kmscore attribute

GeoServer makes the determination on whether to render a layer as raster or vector based on how many features are in the data set and an attribute called `kmscore`. The `kmscore` attribute determines the maximum amount of vector features rendered. It is calculated by this formula:

```
maximum number of features = 10^(kmscore/15)
```

The following table shows the values of this threashold for various values of the `kmscore` parameter:


Washington
Montana
North Dakota
Michigan
Oregon
Idaho
Minnesota
Maine
Vermont
South Dakota
Wisconsin
New York
Wyoming
Connecticut
Nebraska
Iowa
Pennsylvania
Nevada
Utah
Ohio
California
Colorado
Illinois
Kansas
Missouri
Kentucky
Arizona
New Mexico
Oklahoma
Arkansas
Tennessee
South Carolina
Mississippi
Georgia
Texas
Louisiana
Florida
Image NASA
Image © 2007 TerraMetrics
Google


WA
MT
ND
OR
ID
MN
SD
WY
WI
MI
NY
IA
NE
PA
NV
UT
IL
IN
OH
CO
KS
MO
KY
AZ
NM
OK
AR
TN
NC
SC
MS
AL
GA
TX
FL
Image NASA
Image © 2007 TerraMetrics
Google

kmscore	Maximum # of features
0	Force overlay/raster output
10	4
20	21
30	100
40	Approx. 450
50	(*default*) Approx. 2150
60	Approx. 10,000
70	Approx. 45,000
80	Approx. 200,000
90	Approx. 1,000,000
100	Force placemark/vector output

The syntax for specifying `kmscore` is:

```
kmscore=<value>
```

where `<value>` is an integer between 0 and 100. For example:

```
http://localhost:8080/geoserver/wms/kml?layers=topp:states&mode=refresh&kmscore=20
```

The `kmscore` attribute will be ignored if using a reflector mode other than `refresh`.

7.2 Web Feature Service (WFS)

This section describes the Web Feature Service (WFS).

7.2.1 WFS settings

This page details the configuration options for WFS in the web administration interface.

Service Metadata

See the section on *Service Metadata*.

Features

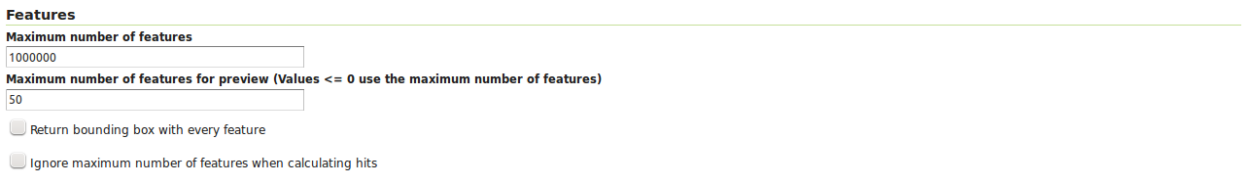


Fig. 7.55: WFS configuration options - Features section

The Open Geospatial Consortium (OGC) Web Feature Service (WFS) is a protocol for serving geographic features across the Web. Feature information that is encoded and transported using WFS includes both

feature geometry and feature attribute values. Basic Web Feature Service (WFS) supports feature query and retrieval. Feature limits and bounding can be configured on the WFS page.

Maximum number of features — Maximum number of features that a WFS GetFeature operation should generate, regardless of the actual number of query hits. A WFS request can potentially contain a large dataset that is impractical to download to a client, and/or too large for a client's renderer. Maximum feature limits are also available for feature types. The default number is 1000000.

Maximum number of features for preview (Values <= 0 use the maximum number of features) - Maximum number of features to use for layer previews. The default is 50 features.

Return bounding box with every feature — When creating the GetFeature GML output, adds an auto-calculated bounds element on each feature type. Not typically enabled, as including bounding box takes up extra bandwidth.

Ignore maximum number of features when calculating hits - When calculating the total number of hits, ignore the Maximum number of features setting. This can be used to get the count of matching features, even if they would not be made available for download because they exceed the maximum count specified. On very large data sets, this can slow down the response.

Service Levels

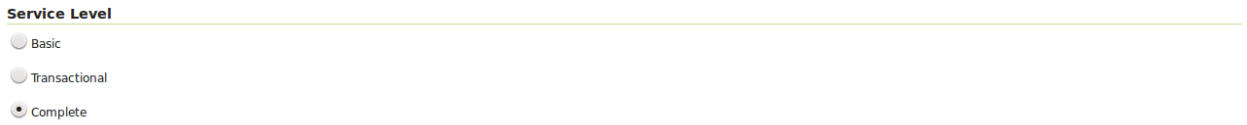


Fig. 7.56: WFS configuration options - Service Level section

GeoServer is compliant with the full "Transactional Web Feature Server" (WFS-T) level of service as defined by the OGC. Specifying the WFS service level limits the capabilities of GeoServer while still remaining compliant. The WFS Service Level setting defines what WFS operations are "turned on".

Basic — Basic service levels provides facilities for searching and retrieving feature data with the GetCapabilities, DescribeFeatureType and GetFeature operations. It is compliant with the OGC basic Web Feature Service. This is considered a READ-ONLY web feature service.

Transactional — In addition to all basic WFS operations, transactional service level supports transaction requests. A transaction request facilitates the creation, deletion, and updating of geographic features in conformance with the OGC Transactional Web Feature Service (WFS-T).

Complete — Includes LockFeature support to the suite of transactional level operations. LockFeature operations help resolve links between related resources by processing lock requests on one or more instances of a feature type.

Configuring additional SRS

WFS 1.1.0 onwards adds the ability to reproject GetFeature output to a user specified target SRS. The list of applicable target SRS is defined on a feature type basis in the capabilities documents, and GeoServer allows reprojection to any supported SRS in its internal database. To declare that GeoServer would have to add 5000+ `otherSRS/otherCRS` elements per feature type, which is clearly impractical. As a result, no declaration is made by default.

A list of values to be declared in all feature types can be provided in the WFS administration panel, as a comma separated list of EPSG codes:

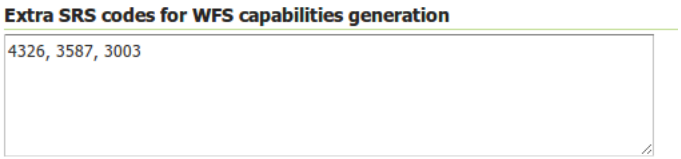


Fig. 7.57: WFS otherSRS/otherCRS configuration

The list will be used only for the capabilities document generation. It does not limit the actual target SRS usage in GetFeature requests.

GML

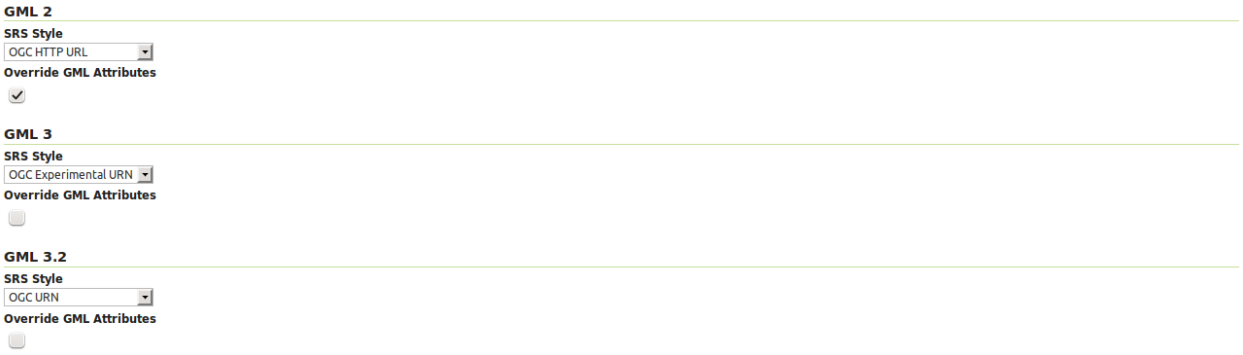


Fig. 7.58: WFS configuration options - GML sections

Geography Markup Language (GML) is the XML-based specification defined by the Open Geospatial Consortium (OGC) to express geographical features. GML serves as a modeling language for geographic systems as well as an open interchange format for geographic transactions on the Internet.

The older GML standard, GML 2 encodes geographic information, including both spatial and non-spatial properties. GML3 extends GML2 support to 3D shapes (surfaces and solids) as well as other advanced facilities. GML 3 is a modular superset of GML 2 that simplifies and minimizes the implementation size by allowing users to select out necessary parts. Additions in GML 3 include support for complex geometries, spatial and temporal reference systems, topology, units of measure, metadata, gridded data, and default styles for feature and coverage visualization. GML 3 is almost entirely backwards compatible with GML 2.

WFS 2.0.0 request return GML 3.2 as the default format, WFS 1.1.0 requests return GML 3 as the default format, and WFS 1.0.0 requests return GML 2 as the default format. For each of the GML formats supported by GeoServer, a different SRS format can be selected.

EPSG Code — Returns the typical EPSG number in the form `EPSG:XXXX` (e.g. `EPSG:4326`). This formats the geographic coordinates in longitude/latitude (x/y) order.

OGC HTTP URL — Returns a URL that identifies each EPSG code: `http://www.opengis.net/gml/srs/epsg.xml#XXXX` (e.g. `http://www.opengis.net/gml/srs/epsg.xml#4326`). This formats the geographic coordinates in longitude/latitude (x/y) order. This format is the default GML 2 SRS convention.

OGC Experimental URN - Returns a URN that identifies each EPSG code: `urn:x-ogc:def:crs:EPSG:XXXX` (e.g. `urn:x-ogc:def:crs:EPSG:4326`). This format was the original GML 3 SRS convention.

OGC URN — (WFS 1.1.1 only) Returns the colon delimited SRS formatting: `urn:ogc:def:crs:EPSG::XXXX` (e.g `urn:ogc:def:crs:EPSG::4326`). This is the revised GML 3 SRS convention, and is the default for GML 3.2. This formats data in the traditional axis order for geographic and cartographic systems—latitude/longitude (y/x).

OGC HTTP URI - Returns a URI that identifies each EPSG code: `http://www.opengis.net/def/crs/EPSG/0/XXXX` (e.g. `http://www.opengis.net/def/crs/EPSG/0/4326`).

For each GML type, there is also an "Override GML Attributes" checkbox. Selecting this (checking the checkbox) will cause attributes to be redefined in the application schema.

Conformance

Fig. 7.59: WFS configuration options - Conformance section

Selecting the *Encode canonical WFS schema location* checkbox modifies the WFS responses to include the canonical schema locations in the `xsi:schemaLocation` attribute, instead of using the default schema locations on the local GeoServer. Note that turning this option on may result in the client not being able to validate the WFS response, depending on network configuration.

Encode response with

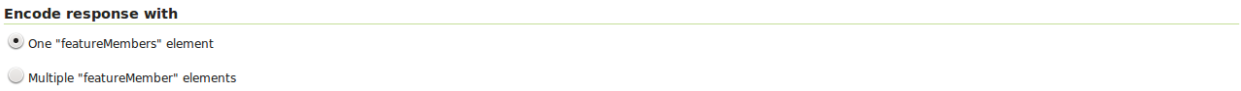


Fig. 7.60: WFS configuration options - Encode response with

The *Encode response with* radio button group has two selection - *One "featureMembers" element* (the default) or *Multiple "featureMember" elements*. This switches the WFS 1.1.0 encoding accordingly. Use of multiple `featureMember` elements may be required for Application Schema referencing.

SHAPE-ZIP output format

Fig. 7.61: WFS configuration options - Encode response with

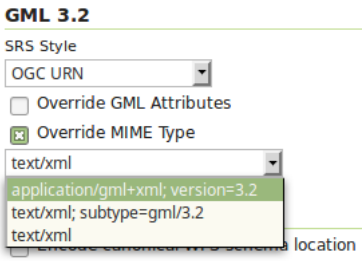
Selecting the *Use ESRI WKT format for SHAPE-ZIP generated .prj files* checkbox modifies how projections are encoded in the Shapefile zip output format. If this checkbox is not selected, OGC WKT format will be used. If this checkbox is selected, ESRI WKT format will be used.

Note: this requires an `esri.properties` file to be provided in the `user_projections` subdirectory of the GeoServer data directory. This may be obtained from the GeoTools EPSG extension.

Override GML 3.2 MIME type

The default MIME used for GML 3.2 encoded responses is `application/gml+xml; version=3.2` which is the MIME type mandated by OGC WFS 2.0 specification. This MIME type is not identified as XML by most common clients like browsers.

Option *Override MIME Type* allows the selection of the MIME type that should be used for the responses encoded in GML 3.2.



The available MIME types are: `application/gml+xml; version=3.2`, `text/xml; subtype=gml/3.2` and `text/xml`.

7.2.2 WFS basics

GeoServer provides support for the Open Geospatial Consortium (OGC) Web Feature Service (WFS) specification, versions **1.0.0**, **1.1.0**, and **2.0.0**. WFS defines a standard for exchanging vector data over the Internet. With a compliant WFS, clients can query both the data structure and the source data. Advanced WFS operations also support feature locking and edit operations.

GeoServer is the reference implementation of all three versions of the standard, completely implementing every part of the protocol. This includes the basic operations of *GetCapabilities*, *DescribeFeatureType*, and *GetFeature*, as well as more advanced options such as *Transaction*. GeoServer WFS is also integrated with its *Security* system to limit access to data and transactions, and supports a variety of *WFS output formats*, making the raw data more widely available.

Differences between WFS versions

The major differences between the WFS versions are:

- WFS 1.1.0 and 2.0.0 return GML3 as the default GML, whereas in WFS 1.0.0, the default is GML2. GML3 adopts marginally different ways of specifying a geometry. GeoServer supports requests in both GML3 and GML2 formats.
- In WFS 1.1.0 and 2.0.0, the SRS (Spatial Reference System, or projection) is specified with `urn:x-ogc:def:crs:EPSG:XXXX`, whereas in WFS 1.0.0 the specification was `http://www.opengis.net/gml/srs/epsg.xml#XXXX`. This change has implications for the *axis order* of the returned data.
- WFS 1.1.0 and 2.0.0 support on-the-fly reprojection of data, which supports returning the data in a SRS other than the native SRS.
- WFS 2.0.0 introduces a new version of the filter encoding specification, adding support for temporal filters.
- WFS 2.0.0 supports joins via a GetFeature request.

- WFS 2.0.0 adds the ability to page results of a GetFeature request via the `startIndex` and `count` parameters. GeoServer now supports this functionality in WFS 1.0.0 and 1.1.0.
- WFS 2.0.0 supports stored queries, which are regular WFS queries stored on the server such that they may be invoked by passing the appropriate identifier with a WFS request.
- WFS 2.0.0 supports SOAP (Simple Object Access Protocol) as an alternative to the OGC interface.

Note: There are also two changes to parameter names which can cause confusion. WFS 2.0.0 uses the `count` parameter to limit the number of features returned rather than the `maxFeatures` parameter used in previous versions. It also changed `typeName` to `typeNames` although GeoServer will accept either.

Axis ordering

WFS 1.0.0 servers return geographic coordinates in longitude/latitude (x/y) order, the most common way to distribute data. For example, most shapefiles adopt this order by default.

However, the traditional axis order for geographic and cartographic systems is the opposite—latitude/longitude (y/x)—and the later WFS specifications respect this. The default axis ordering support is:

- Latitude/longitude—WFS 1.1.0 and WFS 2.0.0
- Longitude/latitude—WFS 1.0.0

This may cause difficulties when switching between servers with different WFS versions, or when upgrading your WFS. To minimize confusion and increase interoperability, GeoServer has adopted the following assumptions when specifying projections in the following formats:

Representation	Assumed axis order
`EPSG:xxxx`	longitude/latitude (x/y)
`http://www.opengis.net/gml/srs/epsg.xml#xxxx`	longitude/latitude (x/y)
`urn:x-ogc:def:crs:EPSG:xxxx`	latitude/longitude (y/x)

7.2.3 WFS reference

The Web Feature Service (WFS) is a standard created by the Open Geospatial Consortium (OGC) for creating, modifying and exchanging vector format geographic information on the Internet using HTTP. A WFS encodes and transfers information in Geography Markup Language (GML), a subset of XML.

The current version of WFS is **2.0.0**. GeoServer supports versions 2.0.0, 1.1.0, and 1.0.0. Although there are some important differences between the versions, the request syntax often remains the same.

A related OGC specification, the *Web Map Service (WMS)*, defines the standard for exchanging geographic information in digital image format.

Benefits of WFS

The WFS standard defines the framework for providing access to, and supporting transactions on, discrete geographic features in a manner that is independent of the underlying data source. Through a combination of discovery, query, locking, and transaction operations, users have access to the source spatial and attribute

data in a manner that allows them to interrogate, style, edit (create, update, and delete), and download individual features. The transactional capabilities of WFS also support the development and deployment of collaborative mapping applications.

Operations

All versions of WFS support these operations:

Operation	Description
`GetCapabilities`	Generates a metadata document describing a WFS service provided by server as well as valid WFS operations and parameters
`DescribeFeatureType`	Returns a description of feature types supported by a WFS service
`GetFeature`	Returns a selection of features from a data source including geometry and attribute values
`LockFeature`	Prevents a feature from being edited through a persistent feature lock
`Transaction`	Edits existing feature types by creating, updating, and deleting

The following operations are available in **version 2.0.0 only**:

Operation	Description
`GetPropertyValue`	Retrieves the value of a feature property or part of the value of a complex feature property from the data store for a set of features identified using a query expression
`GetFeatureWithLock`	Returns a selection of features and also applies a lock on those features
`CreateStoredQuery`	Create a stored query on the WFS server
`DropStoredQuery`	Deletes a stored query from the WFS server
`ListStoredQueries`	Returns a list of the stored queries on a WFS server
`DescribeStoredQueries`	Returns a metadata document describing the stored queries on a WFS server

The following operations are available in **version 1.1.0 only**:

Operation	Description
`GetGMLObject`	Retrieves features and elements by ID from a WFS

Note: In the examples that follow, the fictional URL `http://example.com/geoserver/wfs` is used for illustration. To test the examples, substitute the address of a valid WFS. Also, although the request would normally be defined on one line with no breaks, breaks are added for clarity in the examples provided.

Exceptions

WFS also supports a number of formats for reporting exceptions. The supported values for exception reporting are:

Format	Syntax	Description
XML	`exceptions=text/xml`	*(default)* XML output
JSON	`exceptions=application/json`	Simple JSON
JSONP	`exceptions=text/javascript`	Return a JsonP in the form: parseResponse(...jsonp...). See *WMS vendor parameters* to change the callback name. Note that this format is disabled by default (See *Global variables affecting WMS*).

GetCapabilities

The **GetCapabilities** operation is a request to a WFS server for a list of the operations and services, or *capabilities*, supported by that server.

To issue a GET request using HTTP:

```
http://example.com/geoserver/wfs?
  service=wfs&
  version=1.1.0&
  request=GetCapabilities
```

The equivalent request using POST:

```
<GetCapabilities
 service="WFS"
 xmlns="http://www.opengis.net/wfs"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.opengis.net/wfs
 http://schemas.opengis.net/wfs/1.1.0/wfs.xsd"/>
```

GET requests are simplest to decode, but the POST requests are equivalent.

The parameters for GetCapabilities are:

Parameter	Required?	Description
`service`	Yes	Service name—Value is `WFS`
`version`	Yes	Service version—Value is the current version number. The full version number must be supplied ("1.1.0", "1.0.0"), not the abbreviated form ("1" or "1.1").
`request`	Yes	Operation name—Value is `GetCapabilities`

Although all of the above parameters are technically required as per the specification, GeoServer will provide default values if any parameters are omitted from a request.

The GetCapabilities response is a lengthy XML document, the format of which is different for each of the supported versions. There are five main components in a GetCapabilities document:

Component	Description
`ServiceIdentification`	Contains basic header information for the request such as the `Title` and `ServiceType`. The `ServiceType` indicates which version(s) of WFS are supported.
`ServiceProvider`	Provides contact information about the company publishing the WFS service, including telephone, website, and email.
`OperationsMetadata`	Describes the operations that the WFS server supports and the parameters for each operation. A WFS server may be configured not to respond to the operations listed above.
`FeatureTypeList`	Lists the feature types published by a WFS server. Feature types are listed in the form `namespace:featuretype`. The default projection of the feature type is also listed, along with the bounding box for the data in the stated projection.
`Filter_Capabilities`	Lists the filters, or expressions, that are available to form query predicates, for example, `SpatialOperators` (such as `Equals`, `Touches`) and `ComparisonOperators` (such as `LessThan`, `GreaterThan`). The filters themselves are not included in the GetCapabilities document.

DescribeFeatureType

DescribeFeatureType requests information about an individual feature type before requesting the actual data. Specifically, the operation will request a list of features and attributes for the given feature type, or list the feature types available.

The parameters for DescribeFeatureType are:

Parameter	Required?	Description
`service`	Yes	Service name—Value is `WFS`
`version`	Yes	Service version—Value is the current version number
`request`	Yes	Operation name—Value is `DescribeFeatureType`
`typeNames`	Yes	Name of the feature type to describe (`typeName` for WFS 1.1.0 and earlier)
`exceptions`	No	Format for reporting exceptions—default value is `application/vnd.ogc.se_xml`
`outputFormat`	No	Defines the scheme description language used to describe feature types

To return a list of feature types, the GET request would be as follows. This request will return the list of feature types, sorted by namespace:

```
http://example.com/geoserver/wfs?
  service=wfs&
  version=2.0.0&
  request=DescribeFeatureType
```

To list information about a specific feature type called `namespace:featuretype`, the GET request would be:

```
http://example.com/geoserver/wfs?
  service=wfs&
  version=2.0.0&
  request=DescribeFeatureType&
  typeNames=namespace:featuretype
```

GetFeature

The **GetFeature** operation returns a selection of features from the data source.

This request will execute a GetFeature request for a given layer `namespace:featuretype`:

```
http://example.com/geoserver/wfs?
  service=wfs&
  version=2.0.0&
  request=GetFeature&
  typeNames=namespace:featuretype
```

Executing this command will return the geometries for all features in given a feature type, potentially a large amount of data. To limit the output, you can restrict the GetFeature request to a single feature by including an additional parameter, `featureID` and providing the ID of a specific feature. In this case, the GET request would be:

```
http://example.com/geoserver/wfs?
  service=wfs&
  version=2.0.0&
  request=GetFeature&
  typeNames=namespace:featuretype&
  featureID=feature
```

If the ID of the feature is unknown but you still want to limit the amount of features returned, use the `count` parameter for WFS 2.0.0 or the `maxFeatures` parameter for earlier WFS versions. In the examples below, `N` represents the number of features to return:

```
http://example.com/geoserver/wfs?
  service=wfs&
  version=2.0.0&
  request=GetFeature&
  typeNames=namespace:featuretype&
  count=N
```

```
http://example.com/geoserver/wfs?
  service=wfs&
  version=1.1.0&
  request=GetFeature&
  typeName=namespace:featuretype&
  maxFeatures=N
```

Exactly which N features will be returned depends in the internal structure of the data. However, you can sort the returned selection based on an attribute value. In the following example, an attribute is included in the request using the `sortBy=attribute` parameter (replace `attribute` with the attribute you wish to sort by):

```
http://example.com/geoserver/wfs?
   service=wfs&
   version=2.0.0&
   request=GetFeature&
   typeNames=namespace:featuretype&
   count=N&
   sortBy=attribute
```

The default sort operation is to sort in ascending order. Some WFS servers require the sort order to be specified, even if an ascending order sort if required. In this case, append a `+A` to the request. Conversely,

add a `+D` to the request to sort in descending order as follows:

```
http://example.com/geoserver/wfs?
  service=wfs&
  version=2.0.0&
  request=GetFeature&
  typeNames=namespace:featuretype&
  count=N&
  sortBy=attribute+D
```

There is no obligation to use `sortBy` with `count` in a GetFeature request, but they can be used together to manage the returned selection of features more effectively.

To restrict a GetFeature request by attribute rather than feature, use the `propertyName` key in the form `propertyName=attribute`. You can specify a single attribute, or multiple attributes separated by commas. To search for a single attribute in all features, the following request would be required:

```
http://example.com/geoserver/wfs?
  service=wfs&
  version=2.0.0&
  request=GetFeature&
  typeNames=namespace:featuretype&
  propertyName=attribute
```

For a single property from just one feature, use both `featureID` and `propertyName`:

```
http://example.com/geoserver/wfs?
  service=wfs&
  version=2.0.0&
  request=GetFeature&
  typeNames=namespace:featuretype&
  featureID=feature&
  propertyName=attribute
```

For more than one property from a single feature, use a comma-separated list of values for `propertyName`:

```
http://example.com/geoserver/wfs?
  service=wfs&
  version=2.0.0&
  request=GetFeature&
  typeNames=namespace:featuretype&
  featureID=feature&
  propertyName=attribute1,attribute2
```

While the above permutations for a GetFeature request focused on non-spatial parameters, it is also possible to query for features based on geometry. While there are limited options available in a GET request for spatial queries (more are available in POST requests using filters), filtering by bounding box (BBOX) is supported.

The BBOX parameter allows you to search for features that are contained (or partially contained) inside a box of user-defined coordinates. The format of the BBOX parameter is `bbox=a1,b1,a2,b2``where ``a1`, `b1`, `a2`, and `b2` represent the coordinate values. The order of coordinates passed to the BBOX parameter depends on the coordinate system used. (This is why the coordinate syntax isn't represented with `x` or `y`.) To specify the coordinate system, append `srsName=CRS` to the WFS request, where `CRS` is the Coordinate Reference System you wish to use.

As for which corners of the bounding box to specify, the only requirement is for a bottom corner (left or right) to be provided first. For example, bottom left and top right, or bottom right and top left.

An example request involving returning features based on bounding box would be in the following format:

```
http://example.com/geoserver/wfs?
  service=wfs&
  version=2.0.0&
  request=GetFeature&
  typeNames=namespace:featuretype&
  srsName=CRS
  bbox=a1,b1,a2,b2
```

LockFeature

A **LockFeature** operation provides a long-term feature locking mechanism to ensure consistency in edit transactions. If one client fetches a feature and makes some changes before submitting it back to the WFS, locks prevent other clients from making any changes to the same feature, ensuring a transaction that can be serialized. If a WFS server supports this operation, it will be reported in the server's GetCapabilities response.

In practice, few clients support this operation.

Transaction

The **Transaction** operation can create, modify, and delete features published by a WFS. Each transaction will consist of zero or more Insert, Update, and Delete elements, with each transaction element performed in order. Every GeoServer transaction is *atomic*, meaning that if any of the elements fail, the transaction is abandoned, and the data is unaltered. A WFS server that supports **transactions** is sometimes known as a WFS-T server. **GeoServer fully supports transactions.**

More information on the syntax of transactions can be found in the WFS specification and in the *GeoServer sample requests*.

GetGMLObject

Note: This operation is valid for **WFS version 1.1.0 only**.

A **GetGMLObject** operation accepts the identifier of a GML object (feature or geometry) and returns that object. This operation is relevant only in situations that require *Complex Features* by allowing clients to extract just a portion of the nested properties of a complex feature. As a result, this operation is not widely used by client applications.

GetPropertyValue

Note: This operation is valid for **WFS version 2.0.0 only**.

A **GetPropertyValue** operation retrieves the value of a feature property, or part of the value of a complex feature property, from a data source for a given set of features identified by a query.

This example retrieves the geographic content only of the features in the `topp:states` layer:

```
http://example.com/geoserver/wfs?
  service=wfs&
  version=2.0.0&
  request=GetPropertyValue&
  typeNames=topp:states&
  valueReference=the_geom
```

The same example in a POST request:

```
<wfs:GetPropertyValue service='WFS' version='2.0.0'
 xmlns:topp='http://www.openplans.org/topp'
 xmlns:fes='http://www.opengis.net/fes/2.0'
 xmlns:wfs='http://www.opengis.net/wfs/2.0'
 valueReference='the_geom'>
  <wfs:Query typeNames='topp:states'/>
</wfs:GetPropertyValue>
```

To retrieve value for a different attribute, alter the `valueReference` parameter.

GetFeatureWithLock

Note: This operation is valid for **WFS version 2.0.0 only**.

A **GetFeatureWithLock** operation is similar to a **GetFeature** operation, except that when the set of features are returned from the WFS server, the features are also locked in anticipation of a subsequent transaction operation.

This POST example retrieves the features of the `topp:states` layer, but in addition locks those features for five minutes.

```
<wfs:GetFeatureWithLock service='WFS' version='2.0.0'
 handle='GetFeatureWithLock-tc1' expiry='5' resultType='results'
 xmlns:topp='http://www.openplans.org/topp'
 xmlns:fes='http://www.opengis.net/fes/2.0'
 xmlns:wfs='http://www.opengis.net/wfs/2.0'
 valueReference='the_geom'>
  <wfs:Query typeNames='topp:states'/>
</wfs:GetFeatureWithLock>
```

To adjust the lock time, alter the `expiry` parameter.

CreateStoredQuery

Note: This operation is valid for **WFS version 2.0.0 only**.

A **CreateStoredQuery** operation creates a stored query on the WFS server. The definition of the stored query is encoded in the `StoredQueryDefinition` parameter and is given an ID for a reference.

This POST example creates a new stored query (called "myStoredQuery") that filters the `topp:states` layer to those features that are within a given area of interest (`${AreaOfInterest}`):

```
<wfs:CreateStoredQuery service='WFS' version='2.0.0'
 xmlns:wfs='http://www.opengis.net/wfs/2.0'
 xmlns:fes='http://www.opengis.org/fes/2.0'
 xmlns:gml='http://www.opengis.net/gml/3.2'
 xmlns:myns='http://www.someserver.com/myns'
 xmlns:topp='http://www.openplans.org/topp'>
  <wfs:StoredQueryDefinition id='myStoredQuery'>
    <wfs:Parameter name='AreaOfInterest' type='gml:Polygon'/>
    <wfs:QueryExpressionText
     returnFeatureTypes='topp:states'
     language='urn:ogc:def:queryLanguage:OGC-WFS::WFS_QueryExpression'
     isPrivate='false'>
      <wfs:Query typeNames='topp:states'>
        <fes:Filter>
          <fes:Within>
            <fes:ValueReference>the_geom</fes:ValueReference>
             ${AreaOfInterest}
          </fes:Within>
        </fes:Filter>
      </wfs:Query>
    </wfs:QueryExpressionText>
  </wfs:StoredQueryDefinition>
</wfs:CreateStoredQuery>
```

DropStoredQuery

Note: This operation is valid for **WFS version 2.0.0 only**.

A **DropStoredQuery** operation drops a stored query previous created by a CreateStoredQuery operation. The request accepts the ID of the query to drop.

This example will drop a stored query with an ID of `myStoredQuery`:

```
http://example.com/geoserver/wfs?
  request=DropStoredQuery&
  storedQuery_Id=myStoredQuery
```

The same example in a POST request:

```
<wfs:DropStoredQuery
 xmlns:wfs='http://www.opengis.net/wfs/2.0'
 service='WFS' id='myStoredQuery'/>
```

ListStoredQueries

Note: This operation is valid for **WFS version 2.0.0 only**.

A **ListStoredQueries** operation returns a list of the stored queries currently maintained by the WFS server.

This example lists all stored queries on the server:

```
http://example.com/geoserver/wfs?
  request=ListStoredQueries&
  service=wfs&
  version=2.0.0
```

The same example in a POST request:

```
<wfs:ListStoredQueries service='WFS'
 version='2.0.0'
 xmlns:wfs='http://www.opengis.net/wfs/2.0'/>
```

DescribeStoredQueries

Note: This operation is valid for **WFS version 2.0.0 only**.

A **DescribeStoredQuery** operation returns detailed metadata about each stored query maintained by the WFS server. A description of an individual query may be requested by providing the ID of the specific query. If no ID is provided, all queries are described.

This example describes the existing stored query with an ID of `urn:ogc:def:query:OGC-WFS::GetFeatureById`:

```
http://example.com/geoserver/wfs?
  request=DescribeStoredQueries&
  storedQuery_Id=urn:ogc:def:query:OGC-WFS::GetFeatureById
```

The same example in a POST request:

```
<wfs:DescribeStoredQueries
 xmlns:wfs='http://www.opengis.net/wfs/2.0'
 service='WFS'>
  <wfs:StoredQueryId>urn:ogc:def:query:OGC-WFS::GetFeatureById</wfs:StoredQueryId>
</wfs:DescribeStoredQueries>
```

7.2.4 WFS output formats

WFS returns features and feature information in a number of formats. The syntax for specifying an output format is:

```
outputFormat=<format>
```

where `<format>` is one of the following options:

Format	Syntax	Notes
GML2	`outputFormat=GML2`	Default option for WFS 1.0.0
GML3	`outputFormat=GML3`	Default option for WFS 1.1.0 and 2.0.0
Shapefile	`outputFormat=shape-zip`	ZIP archive will be generated containing the shapefile (see *Shapefile output* below).
JSON	`outputFormat=application/json`	Returns a GeoJSON or a JSON output. Note `outputFormat=json` is only supported for getFeature (for backward compatibility).
JSONP	`outputFormat=text/javascript`	Returns a JSONP in the form: `parseResponse(...json...)`. See *WMS vendor parameters* to change the callback name. Note that this format is disabled by default (See *Global variables affecting WMS*).
CSV	`outputFormat=csv`	Returns a CSV (comma-separated values) file

Note: Some additional output formats (such as *Excel*) are available with the use of an extension. The full list of output formats supported by a particular GeoServer instance can be found by performing a WFS *GetCapabilities* request.

GeoServer provides the `format_options` vendor-specific parameter to specify parameters that are specific to each format. The syntax is:

```
format_options=param1:value1;param2:value2;...
```

Shapefile output

The shapefile format has a number of limitations that would prevent turning data sources into an equivalent shapefile. In order to abide with such limitations the shape-zip output format will automatically apply some transformations on the source data, and eventually split the single colleciton into multiple shapefiles. In particular, the shape-zip format will:

- Reduce attribute names to the DBF accepted length, making sure there are not conflicts (counters being added at the end of the attribute name to handle this).
- Fan out multiple geometry type into parallel shapefiles, named after the original feature type, plus the geometry type as a suffix.
- Fan out multiple shapefiles in case the maximum size is reached

The default max size for both .shp and .dbf file is 2GB, it's possible to modify those limits by setting the GS_SHP_MAX_SIZE and GS_DBF_MAX_SIZE system variables to a different value (as a byte count, the default value being 2147483647).

Shapefile output `format_options`:

- `format_option=filename:<zipfile>`: if a file name is provided, the name is used as the output file name. For example, `format_options=filename:roads.zip`.

Shapefile filename customization

If a file name is not specified, the output file name is inferred from the requested feature type name. The shapefile output format output can be customized by preparing a *Freemarker template* which will configure the file name of the archive (ZIP file) and the files it contains. The default template is:

```
zip=${typename}
shp=${typename}${geometryType}
txt=wfsrequest
```

The `zip` property is the name of the archive, the `shp` property is the name of the shapefile for a given feature type, and `txt` is the dump of the actual WFS request.

The properties available in the template are:

- `typename`—Feature type name (for the `zip` property this will be the first feature type if the request contains many feature types)
- `geometryType`—Type of geometry contained in the shapefile. This is only used if the output geometry type is generic and the various geometries are stored in one shapefile per type.
- `workspace`—Workspace of the feature type
- `timestamp`—Date object with the request timestamp
- `iso_timestamp`—String (ISO timestamp of the request at GMT) in `yyyyMMdd_HHmmss` format

JSON and JSONP output

The JSON output format (and JSONP if enabled) return feature content as a GeoJSON document. Here is an example of a simple GeoJSON file;

```
{   "type": "Feature",
    "geometry": {
       "type": "Point",
       "coordinates": [125.6, 10.1]
    },
    "properties": {
       "name": "Dinagat Islands"
    }
}
```

The output properties can include the use of lists and maps:

```
{
  "type": "Feature",
  "id": "example.3",
  "geometry": {
    "type": "POINT",
    "coordinates": [ -75.70742, 38.557476 ],
  },
  "geometry_name": "geom",
  "properties": {
    "CONDITION": "Orange",
    "RANGE": {"min":"37","max":"93"}
  }
}
```

JSON output `format_options`:

- `format_options=id_policy:<attribute name>=<attribute|true|false>` is used to determine if the id values are included in the output.

  Use `format_options=id_policy:reference_no` for feature id generation using the reference_no attribute, or `format_options=id_policy:reference_no=true` for de-

fault feature id generation, or `format_options=id_policy:reference_no=false` to suppress feature id output.

If id_policy is not specified the geotools default feature id generation is used.

- `format_options=callback:<parseResponse>` applies only to the JSONP output format. See *WMS vendor parameters* to change the callback name. Note that this format is disabled by default (See *Global variables affecting WMS*).

7.2.5 WFS vendor parameters

WFS vendor parameters are non-standard request parameters defined by an implementation to provide enhanced capabilities. GeoServer supports a variety of vendor-specific WFS parameters.

CQL filters

In WFS *GetFeature* GET requests, the `cql_filter` parameter can be used to specify a filter in ECQL (Extended Common Query Language) format. ECQL provides a more compact and readable syntax compared to OGC XML filters.

For full details see the *ECQL Reference* and *CQL and ECQL* tutorial.

The following example illustrates a GET request OGC filter:

```
filter=%3CFilter%20xmlns:gml=%22http://www.opengis.net/gml%22%3E%3CIntersects%3E
↪%3CPropertyName%3Ethe_geom%3C/PropertyName%3E%3Cgml:Point%20srsName=%224326%22%3E
↪%3Cgml:coordinates%3E-74.817265,40.5296504%3C/gml:coordinates%3E%3C/gml:Point%3E%3C/
↪Intersects%3E%3C/Filter%3E
```

Using ECQL, the identical filter would be defined as follows:

```
cql_filter=INTERSECTS(the_geom,%20POINT%20(-74.817265%2040.5296504))
```

Format options

The `format_options` parameter is a container for other parameters that are format-specific. The syntax is:

```
format_options=param1:value1;param2:value2;...
```

The supported format option is:

- `callback` (default is `parseResponse`)—Specifies the callback function name for the JSONP response format
- `id_policy` (default is `true`)- Specifies id generation for the JSON output format. To include feature id in output use an attribute name, or use `true` for feature id generation. To avoid the use of feature id completely use `false`.

Reprojection

As WFS 1.1.0 and 2.0.0 both support data reprojection, GeoServer can store the data in one projection and return GML in another projection. While not part of the specification, GeoServer supports this using WFS 1.0.0 as well. When submitting a WFS *GetFeature* GET request, you can add this parameter to specify the reprojection SRS as follows:

```
srsName=<srsName>
```

The code for the projection is represented by `<srsName>`, for example `EPSG:4326`. For POST requests, you can add the same code to the `Query` element.

XML request validation

GeoServer is less strict than the WFS specification when it comes to the validity of an XML request. To force incoming XML requests to be valid, use the following parameter:

```
strict=[true|false]
```

The default option for this parameter is `false`.

For example, the following request is invalid:

```
<wfs:GetFeature service="WFS" version="1.0.0"
 xmlns:wfs="http://www.opengis.net/wfs">
  <Query typeName="topp:states"/>
</wfs:GetFeature>
```

The request is invalid for two reasons:

- The `Query` element should be prefixed with `wfs:`.
- The namespace prefix has not been mapped to a namespace URI.

That said, the request would still be processed by default. Executing the above command with the `strict=true` parameter, however, would result in an error. The correct syntax should be:

```
<wfs:GetFeature service="WFS" version="1.0.0"
 xmlns:wfs="http://www.opengis.net/wfs"
 xmlns:topp="http://www.openplans.org/topp">
  <wfs:Query typeName="topp:states"/>
</wfs:GetFeature>
```

GetCapabilities namespace filter

WFS *GetCapabilities* requests may be filtered to return only those layers that correspond to a particular namespace by adding the `<namespace>` parameter to the request.

Note: This parameter only affects GetCapabilities requests.

To apply this filter, add the following code to your request:

```
namespace=<namespace>
```

Although providing an invalid namespace will not result in any errors, the GetCapabilities document returned will not contain any layer information.

Warning: Using this parameter may result your GetCapabilities document becoming invalid, as the WFS specification requires the document to return at least one layer.

Note: This filter is related to *Virtual Services*.

7.2.6 WFS schema mapping

One of the functions of the GeoServer WFS is to automatically map the internal schema of a dataset to a feature type schema. This mapping is performed according to the following rules:

- The name of the feature element maps to the name of the dataset.
- The name of the feature type maps to the name of the dataset with the string "Type" appended to it.
- The name of each attribute of the dataset maps to the name of an element particle contained in the feature type.
- The type of each attribute of the dataset maps to the appropriate XML schema type (`xsd:int`, `xsd:double`, and so on).

For example, a dataset has the following schema:

```
myDataset(intProperty:Integer, stringProperty:String, floatProperty:Float,␣
↪geometry:Point)
```

This schema would be mapped to the following XML schema, available via a `DescribeFeatureType` request for the `topp:myDataset` type:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:gml="http://www.opengis.net/gml"
 xmlns:topp="http://www.openplans.org/topp"
 targetNamespace="http://www.openplans.org/topp"
 elementFormDefault="qualified">

  <xsd:import namespace="http://www.opengis.net/gml"
   schemaLocation="http://localhost:8080/geoserver/schemas/gml/3.1.1/base/gml.xsd"/>

  <xsd:complexType name="myDatasetType">
    <xsd:complexContent>
      <xsd:extension base="gml:AbstractFeatureType">
        <xsd:sequence>
          <xsd:element maxOccurs="1" minOccurs="0" name="intProperty" nillable="true"␣
↪type="xsd:int"/>
          <xsd:element maxOccurs="1" minOccurs="0" name="stringProperty" nillable=
↪"true" type="xsd:string"/>
          <xsd:element maxOccurs="1" minOccurs="0" name="floatProperty" nillable="true
↪" type="xsd:double"/>
          <xsd:element maxOccurs="1" minOccurs="0" name="geometry" nillable="true"␣
↪type="gml:PointPropertyType"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:element name="myDataset" substitutionGroup="gml:_Feature" type=
↪"topp:myDatasetType"/>

</xsd:schema>
```

Schema customization

The GeoServer WFS supports a limited amount of schema output customization. A custom schema may be useful for the following:

- Limiting the attributes which are exposed in the feature type schema
- *Changing* the types of attributes in the schema
- Changing the structure of the schema (for example, changing the base feature type)

For example, it may be useful to limit the exposed attributes in the example dataset described above. Start by retrieving the default output as a benchmark of the complete schema. With the feature type schema listed above, the `GetFeature` request would be as follows:

```
<topp:myDataset gml:id="myDataset.1">
 <topp:intProperty>1</topp:intProperty>
  <topp:stringProperty>one</topp:stringProperty>
  <topp:floatProperty>1.1</topp:floatProperty>
  <topp:geometry>
    <gml:Point srsName="urn:x-ogc:def:crs:EPSG:4326">
      <gml:pos>1.0 1.0</gml:pos>
    </gml:Point>
  </topp:geometry>
</topp:myDataset>
```

To remove `floatProperty` from the list of attributes, the following steps would be required:

1. The original schema is modified to remove the `floatProperty`, resulting in the following type definition:

   ```
   <xsd:complexType name="myDatasetType">
     <xsd:complexContent>
       <xsd:extension base="gml:AbstractFeatureType">
         <xsd:sequence>
           <xsd:element maxOccurs="1" minOccurs="0" name="intProperty" nillable="true
   ↪" type="xsd:int"/>
           <xsd:element maxOccurs="1" minOccurs="0" name="stringProperty" nillable=
   ↪"true" type="xsd:string"/>
           <!-- remove the floatProperty element
           <xsd:element maxOccurs="1" minOccurs="0" name="floatProperty" nillable=
   ↪"true" type="xsd:double"/>
           -->
           <xsd:element maxOccurs="1" minOccurs="0" name="geometry" nillable="true"␣
   ↪type="gml:PointPropertyType"/>
         </xsd:sequence>
       </xsd:extension>
       </xsd:complexContent>
   </xsd:complexType>
   ```

2. The modification is saved in a file named `schema.xsd`.
3. The `schema.xsd` file is copied into the feature type directory for the `topp:myDataset` which is:

   ```
   $GEOSERVER_DATA_DIR/workspaces/<workspace>/<datastore>/myDataset/
   ```

   where `<workspace>` is the name of the workspace containing your data store and `<datastore>` is the name of the data store which contains `myDataset`

The modified schema will only be available to GeoServer when the configuration is reloaded or GeoServer is restarted.

A subsequent `DescribeFeatureType` request for `topp:myDataset` confirms the `floatProperty` element is absent:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:gml="http://www.opengis.net/gml"
 xmlns:topp="http://www.openplans.org/topp"
 targetNamespace="http://www.openplans.org/topp"
 elementFormDefault="qualified">

  <xsd:import namespace="http://www.opengis.net/gml"
   schemaLocation="http://localhost:8080/geoserver/schemas/gml/3.1.1/base/gml.xsd"/>

  <xsd:complexType name="myDatasetType">
    <xsd:complexContent>
      <xsd:extension base="gml:AbstractFeatureType">
        <xsd:sequence>
          <xsd:element maxOccurs="1" minOccurs="0" name="intProperty" nillable="true"␣
↪type="xsd:int"/>
          <xsd:element maxOccurs="1" minOccurs="0" name="stringProperty" nillable=
↪"true" type="xsd:string"/>
          <xsd:element maxOccurs="1" minOccurs="0" name="geometry" nillable="true"␣
↪type="gml:PointPropertyType"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:element name="myDataset" substitutionGroup="gml:_Feature" type=
↪"topp:myDatasetType"/>

</xsd:schema>
```

A `GetFeature` request will now return features that don't include the `floatProperty` attribute:

```
<topp:myDataset gml:id="myDataset.1">
  <topp:intProperty>1</topp:intProperty>
  <topp:stringProperty>one</topp:stringProperty>
  <topp:geometry>
    <gml:Point srsName="urn:x-ogc:def:crs:EPSG:4326">
      <gml:pos>1.0 1.0</gml:pos>
    </gml:Point>
  </topp:geometry>
</topp:myDataset>
```

Type changing

Schema customization may be used to perform some **type changing**, although this is limited by the fact that a changed type must be in the same *domain* as the original type. For example, integer types must be changed to integer types, temporal types to temporal types, and so on.

The most common change type requirement is for geometry attributes. In many cases the underlying data set does not have the necessary metadata to report the specific geometry type of a geometry attribute. The automatic schema mapping would result in an element definition similar to the following:

```
<xsd:element maxOccurs="1" minOccurs="0" name="geometry" nillable="true" type=
↪"gml:GeometryPropertyType"/>
```

However if the specific type of the geometry is known, the element definition above could be altered. For point geometry, the element definition could be altered to :

```
<xsd:element maxOccurs="1" minOccurs="0" name="geometry" nillable="true" type=
↪"gml:PointPropertyType"/>
```

7.3 Web Coverage Service (WCS)

This section describes the Web Coverage Service (WCS).

7.3.1 WCS settings

This page details the configuration options for WCS in the web administration interface.

The Web Coverage Service (WCS) provides few options for changing coverage functionality. While various elements can be configured for WFS and WMS requests, WCS allows only metadata information to be edited. This metadata information, entitled *Service Metadata*, is common to WCS, WFS and WMS requests.

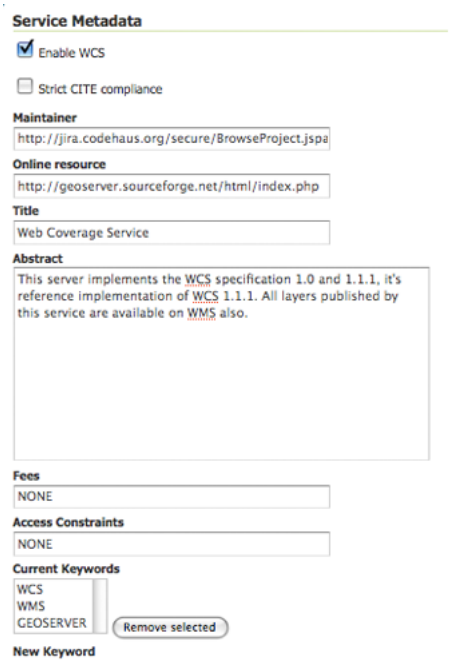


Fig. 7.62: WCS Configuration page

Service Metadata

WCS, WFS, and WMS use common metadata definitions. These nine elements are described in the following table. Though these field types are the same regardless of service, their values are not shared. As such, parameter definitions below refer to the respective service. For example, "Enable" on the WFS Service page, enables WFS service requests and has no effect on WCS or WMS requests.

Field	Description
Enabled	Specifies whether the respective services–WCS, WFS or WMS–should be enabled or disabled. When disabled, the respective service requests will not be processed.
Strict CITE compliance	When selected, enforces strict OGC Compliance and Interoperability Testing Initiative (CITE) conformance. Recommended for use when running conformance tests.
Maintainer	Name of the maintaining body
Online Resource	Defines the top-level HTTP URL of the service. Typically the Online Resource is the URL of the service "home page." (Required) |
Title	A human-readable title to briefly identify this service in menus to clients (required)
Abstract	Provides a descriptive narrative with more information about the service
Fees	Indicates any fees imposed by the service provider for usage of the service. The keyword NONE is reserved to mean no fees and fits most cases.
Access Constraints	Describes any constraints imposed by the service provider on the service. The keyword NONE is reserved to indicate no access constraints are imposed and fits most cases.
Keywords	List of short words associated with the service to aid in cataloging and searching

7.3.2 WCS basics

GeoServer provides support for Open Geospatial Consortium (OGC) Web Coverage Service (WCS) versions 1.0, 1.1 and 2.0. One can think of WCS as the equivalent of *Web Feature Service (WFS)*, but for raster data instead of vector data. It lets you get at the raw coverage information, not just the image. GeoServer is the reference implementation for WCS 1.1.

7.3.3 WCS reference

Introduction

The Web Coverage Service (WCS) is a standard created by the OGC that refers to the receiving of geospatial information as 'coverages': digital geospatial information representing space-varying phenomena. One can think of it as *Web Feature Service (WFS)* for *raster* data. It gets the 'source code' of the map, but in this case its not raw vectors but raw imagery.

An important distinction must be made between WCS and *Web Map Service (WMS)*. They are similar, and can return similar formats, but a WCS is able to return more information, including valuable metadata and more formats. It additionally allows more precise queries, potentially against multi-dimensional backend formats.

Benefits of WCS

WCS provides a standard interface for how to request the raster source of a geospatial image. While a WMS can return an image it is generally only useful as an image. The results of a WCS can be used for complex modeling and analysis, as it often contains more information. It also allows more complex querying - clients can extract just the portion of the coverage that they need.

Operations

WCS can perform the following operations:

Operation	Description
`GetCapabilities`	Retrieves a list of the server's data, as well as valid WCS operations and parameters
`DescribeCoverage`	Retrieves an XML document that fully describes the request coverages.
`GetCoverage`	Returns a coverage in a well known format. Like a WMS GetMap request, but with several extensions to support the retrieval of coverages.

Note: The following examples show the 1.1 protocol, the full specification for versions 1.0, 1.1 and 2.0 are available on the OGC web site

GetCapabilities

The **GetCapabilities** operation is a request to a WCS server for a list of what operations and services ("capabilities") are being offered by that server.

A typical GetCapabilities request would look like this (at URL `http://www.example.com/wcs`):

Using a GET request (standard HTTP):

```
http://www.example.com/wcs?
service=wcs&
AcceptVersions=1.1.0&
request=GetCapabilities
```

Here there are three parameters being passed to our WCS server, `service=wcs`, `AcceptVersions=1.1.0`, and `request=GetCapabilities`. At a bare minimum, it is required that a WCS request have the service and request parameters. GeoServer relaxes these requirements (setting the default version if omitted), but "officially" they are mandatory, so they should always be included. The *service* key tells the WCS server that a WCS request is forthcoming. The *AcceptsVersion* key refers to which version of WCS is being requested. The *request* key is where the actual GetCapabilities operation is specified.

WCS additionally supports the Sections parameter that lets a client only request a specific section of the Capabilities Document.

DescribeCoverage

The purpose of the **DescribeCoverage** request is to additional information about a Coverage a client wants to query. It returns information about the crs, the metadata, the domain, the range and the formats it is available in. A client generally will need to issue a DescribeCoverage request before being sure it can make the proper GetCoverage request.

GetCoverage

The **GetCoverage** operation requests the actual spatial data. It can retrieve subsets of coverages, and the result can be either the coverage itself or a reference to it. The most powerful thing about a GetCoverage request is its ability to subset domains (height and time) and ranges. It can also do resampling, encode in different data formats, and return the resulting file in different ways.

7.3.4 WCS output formats

WCS output formats are configured coverage by coverage. The current list of output formats follows:

Images:

- JPEG - (format=jpeg)
- GIF - (format=gif)
- PNG - (format=png)
- Tiff - (format=tif)
- BMP - (format=bmp)

Georeferenced formats:

- GeoTiff - (format=geotiff)
- GTopo30 - (format=gtopo30)
- ArcGrid - (format=ArcGrid)
- GZipped ArcGrid - (format=ArcGrid-GZIP)

Beware, in the case of ArcGrid, the GetCoverage request must make sure the x and y resolution are equal, otherwise an exception will be thrown (ArcGrid is designed to have square cells).

7.3.5 WCS Vendor Parameters

namespace

Requests to the WCS GetCapabilities operation can be filtered to only return layers corresponding to a particular namespace.

Sample code:

```
http://example.com/geoserver/wcs?
   service=wcs&
   version=1.0.0&
   request=GetCapabilities&
   namespace=topp
```

Using an invalid namespace prefix will not cause any errors, but the document returned will not contain information on any layers.

cql_filter

The `cql_filter` parameter is similar to same named WMS parameter, and allows expressing a filter using ECQL (Extended Common Query Language). The filter is sent down into readers exposing a `Filter` read parameter.

For example, assume a image mosaic has a tile index with a `cloudCover` percentage attribute, then it's possible to mosaic only granules with a cloud cover less than 10% using:

```
cql_filter=cloudCover < 10
```

For full details see the *ECQL Reference* and *CQL and ECQL* tutorial.

sortBy

The `sortBy` parameter allows to control the order of granules being mosaicked, using the same syntax as WFS 1.0, that is:

- `&sortBy=att1 A|D,att2 A|D, ...`

This maps to a "SORTING" read parameter that the coverage reader might expose (image mosaic exposes such parameter).

In image mosaic, this causes the first granule found in the sorting will display on top, and then the others will follow.

Thus, to sort a scattered mosaic of satellite images so that the most recent image shows on top, and assuming the time attribute is called `ingestion` in the mosaic index, the specification will be `&sortBy=ingestion D`.

7.3.6 WCS configuration

Coverage processing

The WCS processing chain can be tuned in respect of how raster overviews and read subsampling are used.

The overview policy has four possible values:

Option	Description	Version
Lower resolution overview	Looks up the two overviews with a resolution closest to the one requested and chooses the one at the lower resolution.	2.0.3
Don't use overviews	Overviews will be ignored, the data at its native resolution will be used instead. This is the default value.	2.0.3
Higher resolution overview	Looks up the two overviews with a resolution closest to the one requested and chooses the one at the higher resolution.	2.0.3
Closest overview	Looks up the overview closest to the one requested	2.0.3

While reading coverage data at a resolution lower than the one available on persistent storage its common to use subsampling, that is, read one every N pixels as a way to reduce the resolution of the data read in memory. **Use subsampling** controls wheter subsampling is enabled or not.

Request limits

The request limit options allow the administrator to limit the resources consumed by each WCS `GetCoverage` request.

The request limits limit the size of the image read from the source and the size of the image returned to the client. Both of these limits are to be considered a worst case scenario and are setup to make sure the server never gets asked to deal with too much data.

Option	Description	Version
Maximum input memory	Sets the maximum amount of memory, in kilobytes, a GetCovearge request might use, at most, to read a coverage from the data source. The memory is computed as `rw * rh * pixelsize`, where `rw` and `rh` are the size of the raster to be read and `pixelsize` is the dimension or a pixel (e.g., a RGBA image will have 32bit pixels, a batimetry might have 16bit signed int ones)	2.0.3
Maximum output memory	Sets the maximum amount of memory, in kilobytes, a GetCoverage request might use, at most, to host the resulting raster. The memory is computed as `ow * oh * pixelsize`, where `ow` and `oh` are the size of the raster to be generated in output.	2.0.3
Max number of dimension values	Sets the maximum number of dimension (time, at least for now) values that a client can request in a GetCoverage request (the work to be done is usually proportional to said number of times, and the list of values is kept in memory during the processing)	2.14.0

To understand the limits let's consider a very simplified examle in which no tiles and overviews enter the game:

- The request hits a certain area of the original raster. Reading it at full resolution requires grabbing a raster of size `rw * rh`, which has a certain number of bands, each with a certain size. The amount of memory used for the read will be `rw * rh * pixelsize`. This is the value measured by the input memory limit
- The WCS performs the necessary processing: band selection, resolution change (downsampling or upsampling), reprojection
- The resuling raster will have size `ow * oh` and will have a certain number of bands, possibly less than the input data, each with a certain size. The amount of memory used for the final raster will be `ow * oh * pixelsize`. This is the value measured by the output memory limit.
- Finally the resulting raster will be encoded in the output format. Depending on the output format structure the size of the result might be higher than the in memory size (ArcGrid case) or smaller (for example in the case of GeoTIFF output, which is normally LZW compressed)

In fact reality is a bit more complicated:

- The input source might be tiled, which means there is no need to fully read in memory the region, but it is sufficient to do so one tile at a time. The input limits won't consider inner tiling when computing the limits, but if all the input coverages are tiled the input limits should be designed considering the amount of data to be read from the persistent storage as opposed to the amount of data to be stored in memory
- The reader might be using overviews or performing subsampling during the read to avoid actually reading all the data at the native resolution should the output be subsampled
- The output format might be tile aware as well (GeoTIFF is), meaning it might be able to write out one tile at a time. In this case not even the output raster will be stored in memory fully at any given time.

Only a few input formats are so badly structure that they force the reader to read the whole input data in one shot, and should be avoided. Examples are: * JPEG or PNG images with world file * Single tiled and JPEG compressed GeoTIFF files

7.3.7 WCS Request Builder

GeoServer includes a request builder for building and testing out WCS requests. Since WCS requests can be cumbersome to author, this tool can make working with WCS much easier.

Accessing the WCS Request Builder

To access the WCS Request Builder:

1. Navigate to the *Web administration interface*.
2. Click the *Demos* link on the left side.
3. Select *WCS Request Builder* from the list of demos.

Fig. 7.63: WCS request builder in the list of demos

Using the WCS Request Builder

The WCS Request Builder consists of a form which can be used to generate a number of different types of requests.

When first opened, the form is short, only including the following options:

- *WCS Version*—Version of WCS to use when crafting the request. Options are *1.0.0* and *1.1.1*.
- *Coverage name*—Coverage to use in the request. Any published (raster) layer in GeoServer can be selected.

> **Note:** All other options displayed will be non-functional until *Coverage name* is selected.

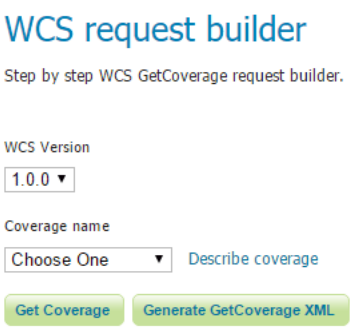


Fig. 7.64: WCS request builder in its initial form

Once selected, the remainder of the form will be displayed. The following options are available:

- *Spatial subset*—Sets the extent of the GetCoverage request in units of the layer CRS. Defaults to the full extent of the layer.
- *Coordinate reference system*—Source CRS of the layer. Default is the CRS of the layer in GeoServer.
- *Specify source grid manually (1.0.0 only)*—If checked, allows for determining the grid of pixels for the output.
- *Target coverage layout (1.1.1 only)*—Specifies how the dimensions of the output grid will be determined:
  - *Automatic target layout*—Sets that the output grid will be determined automatically.
  - *Specify grid resolutions*—Sets the resolution of the output grid. X and Y resolutions can be set differently.
  - *Specify "grid to world" transformation*—Sets the output using latitude/longitude, as well as X and Y scale and shear values.
- *Target CRS*—CRS of the result (output) of the GetCoverage request. If different from the *Coordinate reference system*, the result will be a reprojection into the target CRS.
- *Output format*—Format of the result (output) of the GetCoverage request. Any valid WCS output format is allowed. Default is *GeoTIFF*.

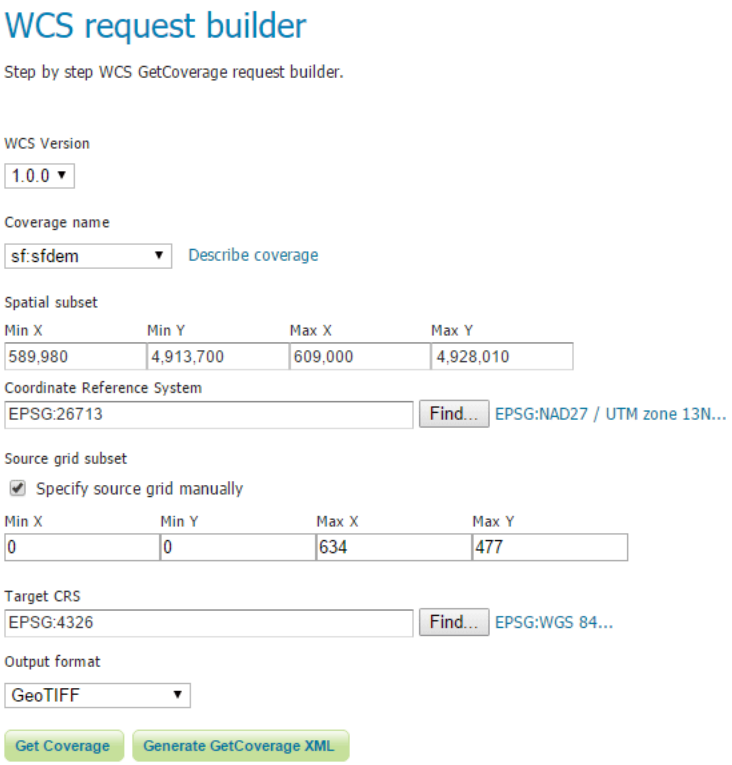


Fig. 7.65: WCS request builder form (WCS version 1.0.0)

There is also a link for *Describe coverage* next to the *Coverage name* which will execute a *WCS DescribeCoverage* request for the particular layer.

At the bottom of the form are two buttons for form submission:

- *Get Coverage*—Executes a GetCoverage request using the parameters in the form. This will usually result in a file which can be downloaded.

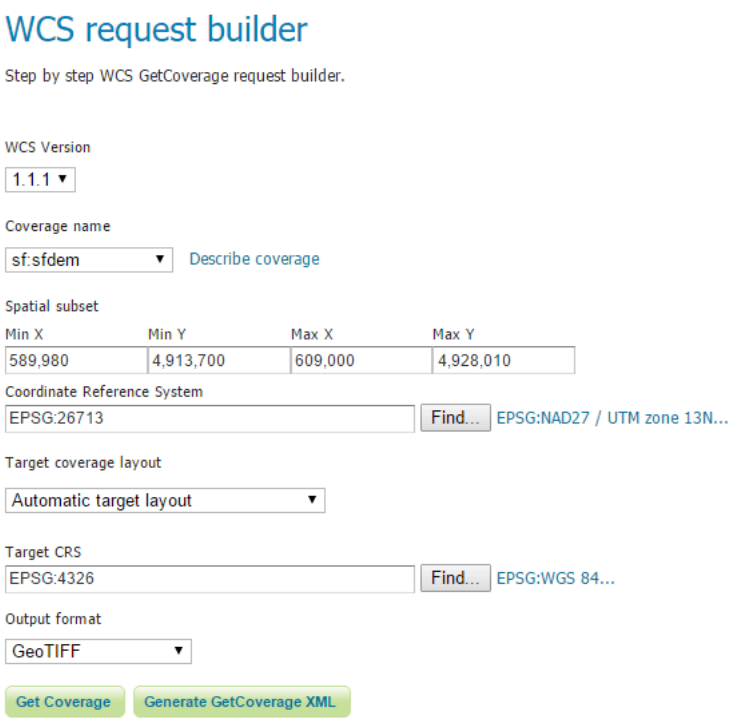


Fig. 7.66: WCS request builder form (WCS version 1.1.1)

- *Generate GetCoverage XML*—Generates the GetCoverage request using the parameters in the form and then, instead of executing it, outputs the request itself to the screen.

7.4 Web Processing Service (WPS)

Web Processing Service (WPS) is an OGC service for the publishing of geospatial processes, algorithms, and calculations. The WPS service is available as an extension for geoserver providing an execute operation for data processing and geospatial analysis.

WPS is not a part of GeoServer by default, but is available as an extension.

The main advantage of GeoServer WPS over a standalone WPS is **direct integration** with other GeoServer services and the data catalog. This means that it is possible to create processes based on data served in GeoServer, as opposed to sending the entire data source in the request. It is also possible for the results of a process to be stored as a new layer in the GeoServer catalog. In this way, WPS acts as a full remote geospatial analysis tool, capable of reading and writing data from and to GeoServer.

For the official WPS specification, see OGC Web Processing Service 05-007r7.

7.4.1 Installing the WPS extension

The WPS module is not a part of GeoServer core, but instead must be installed as an extension. To install WPS:

1. Navigate to the GeoServer download page
2. Find the page that matches the exact version of GeoServer you are running.

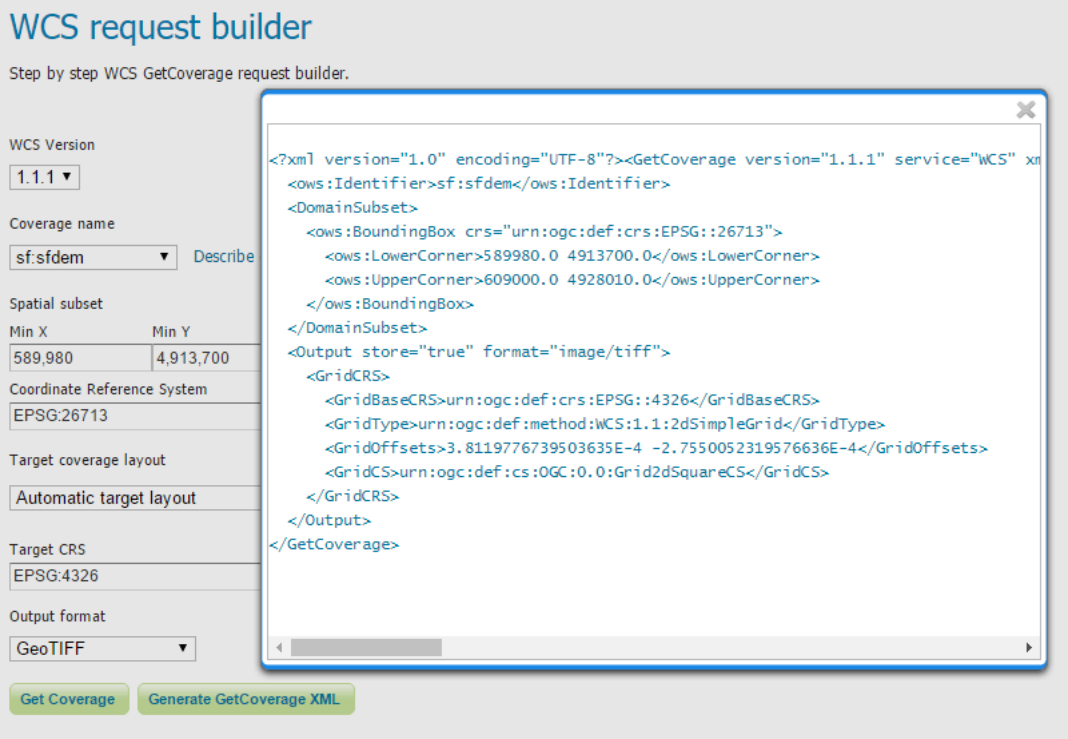


Fig. 7.67: WCS request builder showing GetCoverage XML

Warning: Be sure to match the version of the extension with that of GeoServer, otherwise errors will occur.

3. Download the WPS extension. The download link for *WPS* will be in the *Extensions* section under *Other*.
4. Extract the files in this archive to the `WEB-INF/lib` directory of your GeoServer installation.
5. Restart GeoServer.

After restarting, load the *Web administration interface*. If the extension loaded properly, you should see an extra entry for WPS in the *Service Capabilities* column. If you don't see this entry, check the logs for errors.

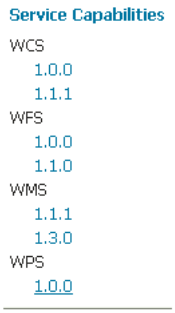


Fig. 7.68: *A link for the WPS capabilities document will display if installed properly*

Configuring WPS

WPS processes are subject to the same feature limit as the WFS service. The limit applies to process **input**, so even processes which summarize data and return few results will be affected if applied to very large datasets. The limit is set on the *WFS settings* Admin page.

Warning: If the limit is encountered during process execution, no error is given. Any results computed by the process may be incomplete

7.4.2 WPS Operations

WPS defines three operations for the discovery and execution of geospatial processes. The operations are:

- GetCapabilities
- DescribeProcess
- Execute

GetCapabilities

The **GetCapabilities** operation requests details of the service offering, including service metadata and metadata describing the available processes. The response is an XML document called the **capabilities document**.

The required parameters, as in all OGC GetCapabilities requests, are `service=WPS`, `version=1.0.0` and `request=GetCapabilities`.

An example of a GetCapabilities request is:

```
http://localhost:8080/geoserver/ows?
  service=WPS&
  version=1.0.0&
  request=GetCapabilities
```

DescribeProcess

The **DescribeProcess** operation requests a description of a WPS process available through the service.

The parameter `identifier` specifies the process to describe. Multiple processes can be requested, separated by commas (for example, `identifier=JTS:buffer,gs:Clip`). At least one process must be specified.

Note: As with all OGC parameters, the keys (`request`, `version`, etc) are case-insensitive, and the values (`GetCapabilities`, `JTS:buffer`, etc.) are case-sensitive. GeoServer is generally more relaxed about case, but it is best to follow the specification.

The response is an XML document containing metadata about each requested process, including the following:

- Process name, title and abstract

- For each input and output parameter: identifier, title, abstract, multiplicity, and supported datatype and format

An example request for the process `JTS:buffer` is:

```
http://localhost:8080/geoserver/ows?
  service=WPS&
  version=1.0.0&
  request=DescribeProcess&
  identifier=JTS:buffer
```

The response XML document contains the following information:

Title	"Buffers a geometry using a certain distance"
Inputs	**geom**: "The geometry to be buffered" *(geometry, mandatory)* **distance**: "The distance (same unit of measure as the geometry)" *(double, mandatory)* **quadrant segments**: "Number of quadrant segments. Use > 0 for round joins, 0 for flat joins, < 0 for mitred joins" *(integer, optional)* **capstyle**: "The buffer cap style, round, flat, square" *(literal value, optional)*
Output formats	One of GML 3.1.1, GML 2.1.2, or WKT

Execute

The **Execute** operation is a request to perform the process with specified input values and required output data items. The request may be made as either a GET URL, or a POST with an XML request document. Because the request has a complex structure, the POST form is more typically used.

The inputs and outputs required for the request depend on the process being executed. GeoServer provides a wide variety of processes to process geometry, features, and coverage data. For more information see the section *Process Cookbook*.

Below is an example of a `Execute` POST request. The example process (`JTS:buffer`) takes as input a geometry `geom` (in this case the point `POINT(0 0)`), a `distance` (with the value `10`), a quantization factor `quadrantSegments` (here set to be 1), and a `capStyle` (specified as `flat`). The `<ResponseForm>` element specifies the format for the single output `result` to be GML 3.1.1.

```
<?xml version="1.0" encoding="UTF-8"?>
<wps:Execute version="1.0.0" service="WPS" xmlns:xsi="http://www.w3.org/2001/
↪XMLSchema-instance" xmlns="http://www.opengis.net/wps/1.0.0" xmlns:wfs="http://www.
↪opengis.net/wfs" xmlns:wps="http://www.opengis.net/wps/1.0.0" xmlns:ows="http://www.
↪opengis.net/ows/1.1" xmlns:gml="http://www.opengis.net/gml" xmlns:ogc="http://www.
↪opengis.net/ogc" xmlns:wcs="http://www.opengis.net/wcs/1.1.1" xmlns:xlink="http://
↪www.w3.org/1999/xlink" xsi:schemaLocation="http://www.opengis.net/wps/1.0.0 http://
↪schemas.opengis.net/wps/1.0.0/wpsAll.xsd">
  <ows:Identifier>JTS:buffer</ows:Identifier>
  <wps:DataInputs>
    <wps:Input>
      <ows:Identifier>geom</ows:Identifier>
      <wps:Data>
        <wps:ComplexData mimeType="application/wkt"><![CDATA[POINT(0 0)]]></
↪wps:ComplexData>
      </wps:Data>
    </wps:Input>
    <wps:Input>
      <ows:Identifier>distance</ows:Identifier>
```

```
      <wps:Data>
        <wps:LiteralData>10</wps:LiteralData>
      </wps:Data>
    </wps:Input>
    <wps:Input>
      <ows:Identifier>quadrantSegments</ows:Identifier>
      <wps:Data>
        <wps:LiteralData>1</wps:LiteralData>
      </wps:Data>
    </wps:Input>
    <wps:Input>
      <ows:Identifier>capStyle</ows:Identifier>
      <wps:Data>
        <wps:LiteralData>flat</wps:LiteralData>
      </wps:Data>
    </wps:Input>
  </wps:DataInputs>
  <wps:ResponseForm>
    <wps:RawDataOutput mimeType="application/gml-3.1.1">
      <ows:Identifier>result</ows:Identifier>
    </wps:RawDataOutput>
  </wps:ResponseForm>
</wps:Execute>
```

The process performs a buffer operation using the supplied inputs, and returns the outputs as specified. The response from the request is (with numbers rounded for clarity):

```
<?xml version="1.0" encoding="utf-8"?>
<gml:Polygon xmlns:sch="http://www.ascc.net/xml/schematron"
 xmlns:gml="http://www.opengis.net/gml"
 xmlns:xlink="http://www.w3.org/1999/xlink">
  <gml:exterior>
    <gml:LinearRing>
      <gml:posList>
        10.0 0.0
        0.0 -10.0
        -10.0 0.0
        0.0 10.0
        10.0 0.0
      </gml:posList>
    </gml:LinearRing>
  </gml:exterior>
</gml:Polygon>
```

For help in generating WPS requests you can use the built-in interactive *WPS Request Builder*.

Dismiss

According to the WPS specification, an asynchronous process execution returns a back link to a status location that the client can ping to get progress report about the process, and eventually retrieve its final results.

In GeoServer this link is implemented as a pseudo-operation called `GetExecutionStatus`, and the link has the following structure:

```
http://host:port/geoserver/ows?service=WPS&version=1.0.0&request=GetExecutionStatus&
↪executionId=397e8cbd-7d51-48c5-ad72-b0fcbe7cfbdb
```

The `executionId` identifies the running request, and can be used in a the `Dismiss` vendor operation in order to cancel the execution of the process:

> http://host:port/geoserver/ows?service=WPS&version=1.0.0&request=Dismiss&executionId=397e8cbd-7d51-48c5-ad72-b0fcbe7cfbdb

Upon receipt GeoServer will do its best to stop the running process, and subsequent calls to `Dismiss` or `GetExecutionStatus` will report that the executionId is not known anymore. Internally, GeoServer will stop any process that attempts to report progress, and poison input and outputs to break the execution of the process, but the execution of processes that already got their inputs, and are not reporting their progress back, will continue until its natural end.

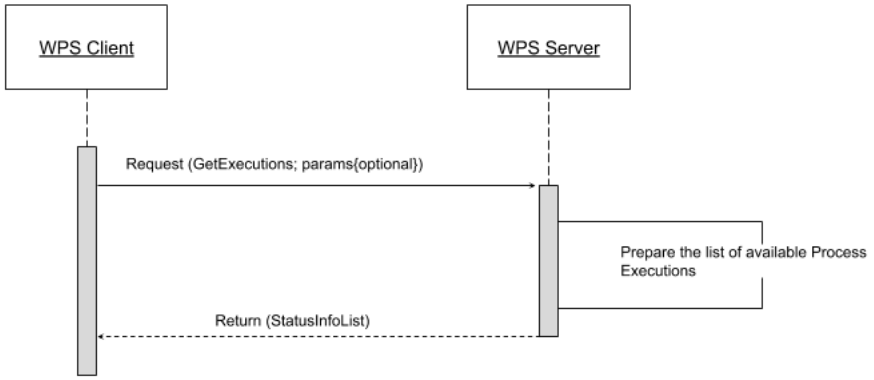
For example, let's consider the "geo:Buffer" process, possibly working against a very large input GML geometry, to be fetched from another host. The process itself does a single call to a JTS function, which cannot report progress. Here are three possible scenarios, depending on when the Dismiss operation is invoked:

- Dismiss is invoked while the GML is being retrieved, in this case the execution will stop immediately
- Dismiss is invoked while the process is doing the buffering, in this case, the execution will stop as soon as the buffering is completed
- Dismiss is invoked while the output GML is being encoded, also in this case the execution will stop immediately

GetExecutions

Note: This is an extension of the GeoServer WPS Service. This operation is specific to this GeoServer instance.

This specific operation allows a client to recognize the list of WPS Executions.



The client makes a simple "GetExecutions" request to the WPS Server, in order to get back an XML document containing the list of current Execution Statuses.

It is also possible to filter the "GetExecutions" request along with simple parameters, in order to refine the output and get back only the executions status we are looking for.

Adding a bit more to this, AUTHORIZATION headers must be sent along with the "GetExecutions" request; the WPS Server will be able, if the security subsystem is available and enable on the latter, to prove the list resources to the client itself.

The operation will return only the list of available Executions the logged in user has started, except in the case it is an Administrator. In that case he will be able to get the whole list.

If the "lineage" option of the WPS Execute Request has been specified, the client will be able to retrieve the Execute Inputs values provided to the process Identifier.

StatusInfo Document

Refers to http://docs.opengeospatial.org/is/14-065/14-065.html 9.5 and extends it.

The StatusInfo document is used to provide identification and status information about jobs on a WPS server. The operation adds additional fields to the StatusInfo Document reporting also the WPS Process Identifier and other information on estimated execution and expiration time.

Names	Definition	Data type and values	Multiplicity and use
JobID	Unambiguously identifier of a execution job within a WPS instance.	Character String [a]	One (mandatory)
Identifier	Unambiguously identifier of a process within a WPS instance.	ows:Identifier	One (mandatory)
Status	Well-known identifier describing the status of the job.	Character String [b]	One (mandatory)
ExpirationDate	Date and time by which the job and its results will be no longer accessible. [c]	ISO-8601 date/time string in the form YYYY-MM-DDTHH:MM:SS.SSSZ with T separator character and Z suffix for coordinated universal time (UTC)	Zero or one (optional) Include if available.
EstimatedCompletion	Date and time by which the processing job will be finished. By default is can be evaluated as "time elapsed / percentage complete" and then roll new extension points in case the process or their factory have a better way.	ISO-8601 date/time string in the form YYYY-MM-DDTHH:MM:SS.SSSZ with T separator character and Z suffix for coordinated universal time (UTC)	Zero or one (optional) Include if available.
NextPoll	Date and time for the next suggested status polling.	ISO-8601 date/time string in the form YYYY-MM-DDTHH:MM:SS.SSSZ with T separator character and Z suffix for coordinated universal time (UTC)	Zero or one (optional) Include if available.
PercentCompleted	Percentage of process that has been completed.	Integer{0..100} [d]	Zero or one (optional) Include if available.

a) Particularly suitable JobIDs are UUIDs or monotonic identifiers such as unique timestamps. If the privacy of a Processing Job is imperative, the JobID should be non-guessable.
b) The basic status set is defined in http://docs.opengeospatial.org/is/14-065/14-065.html Table 3. Additional states may be defined by certain operations or extensions of this standard.
c) This element will usually become available when the execution has finished (Status = "finished").
d) Zero (0) means the execution has just started, and 100 means the job is complete. This value is informative only without any accuracy guarantees.

Table 1 – StatusInfo structure

GetExecutionsOperation

The GetExecutions Operation allows WPS clients to retrieve the list of available process jobs running on a WPS instance. The output is returned in the form of an XML document.

The GetExecutions Operation returns only the list of available Executions the logged in user has started, except in the case it is an Administrator. In that case he will be able to get the whole list.

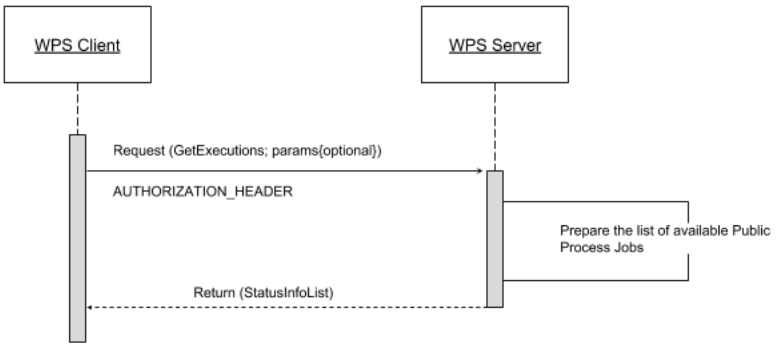


Figure 1 - GetExecutionsOperation UML sequence diagram

GetExecutionsRequest

The GetExecutions Request is a common structure for synchronous execution. It inherits basic properties from the RequestBaseType and contains additional elements that allow to filter out, refine and order the list of available Process Jobs.

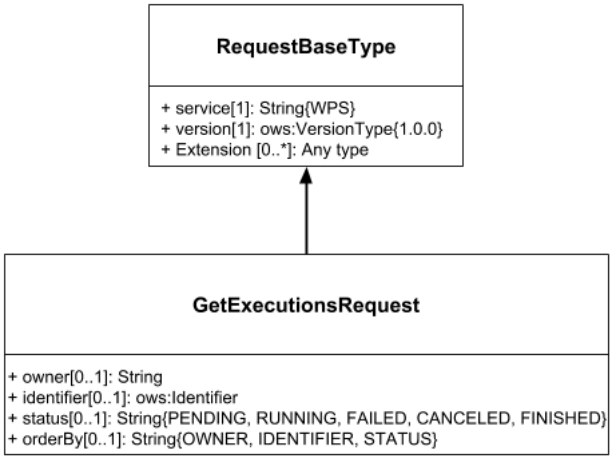


Figure 2 – GetExecutionsRequest UML class diagram

GetExecutionsResponse

The GetExecutionsResponse it is always in the form of an XML document. Except in case of Exception, the response document will contain a list of StatusInfo elements filtered, refined or ordered accordingly to the specified parameters.

Response paging

Response paging is the ability of a client to scroll through a set of response values, N values at-a-time much like one scrolls through the response from a search engine one page at a time.

Similarly to the WFS 2.0.0 response paging mechanism (see See section "7.7.4.4 Response paging" of the specification), the output will show to the client the following attributes as part of the response document.

Names	Definition	Data type and values	Multiplicity and use
Owner	Unambiguous identifier of a user within a WPS instance. Filters out all the jobs not anonymous or not belonging to the specified user.	Character String [a]	Zero or one (optional).
Identifier	Unambiguous identifier of a process within a WPS instance. Filters out all the jobs not belonging to the specified process identifier.	ows:Identifier Value shall be one of the process identifiers listed in the ProcessSummary elements in the Capabilities document.	Zero or one (optional).
Status	Well-known identifier describing the status of the job. Filters out all the jobs with an Execution Status different from the one specified.	String{PENDING, RUNNING, FAILED, CANCELED, FINISHED} [b]	Zero or one (optional).
OrderBy	One of the OWNER, IDENTIFIER, STATUS. The Response Document will be ordered accordingly to the value specified. If not value has been specified, the list of Processed will be ordered by Identifier and ExecutionDate.	String{OWNER, IDENTIFIER, STATUS}	Zero or one (optional).

a) It must match one of the usernames available on the Server, if any available. GetStatusesListResponse contents may vary accordingly to security constraints.
b) The basic status set is defined in http://docs.opengeospatial.org/is/14-065/14-065.html Table 3. Additional states may be defined by certain operations or extensions of this standard.

Table 2 – GetExecutionsRequest structure

Names	Definition	Data type and values	Multiplicity and use
count	The number of values presented on one page retrieved by the GetExecution Response. This value does not represent the total number of values, but the number of elements contained into the current page.	Integer representing number of values presented on one page. This can be <= max items per page.	One (mandatory). (>= 0 && <= maxPageItems)
next	URI of the next page. If absent, the current page is the last page available.	String representing the URI of the next page.	Zero or one (optional).
previous	URI of the previous page. If absent, the current page is the first page available.	String representing the URI of the previous page.	Zero or one (optional).

GetExecutionsExceptions

When a WPS server encounters an error while performing an GetExecutionsResponse, it shall return an exception report as specified in clause 8 of [OGC 06-121r9]. If appropriate, the server shall use additional exception codes as defined in this section.

exceptionCode value	ExceptionText	locator	HTTP status code
DataNotAccessible	One of the referenced input data sets was inaccessible.	List of violating input identifiers	400 (Bad request)
NoSuchProcess	One of the identifiers passed does not match with any of the processes offered by this server.	List of violating process identifiers.	400 (Bad request)
NoSuchParameter	A wrong parameter value has been specified for enumerated ones.	List of violating process parameters.	400 (Bad request)
InternalServerError	None (omit exception text)	None (omit locator parameter)	500 (Internal Server Error) [a]
a) If the cause could not be determined or is not covered by the above exception codes, this exception shall be used.			

Table 3 – GetExecutionsException codes

Retrieve the WPS Execute Input values

The GetExecutions Operations tries (best-effort) to retrieve the Input values specified from the Execute Request **iff** the `lineage` option has been provided to the Execute Request.

Example requests with the `lineage` option active

```
<?xml version="1.0" encoding="UTF-8"?>
<wps:Execute version="1.0.0" service="WPS" xmlns:xsi="http://www.w3.org/2001/
↪XMLSchema-instance" xmlns="http://www.opengis.net/wps/1.0.0" xmlns:wfs="http://www.
↪opengis.net/wfs" xmlns:wps="http://www.opengis.net/wps/1.0.0" xmlns:ows="http://www.
↪opengis.net/ows/1.1" xmlns:gml="http://www.opengis.net/gml" xmlns:ogc="http://www.
↪opengis.net/ogc" xmlns:wcs="http://www.opengis.net/wcs/1.1.1" xmlns:xlink="http://
↪www.w3.org/1999/xlink" xsi:schemaLocation="http://www.opengis.net/wps/1.0.0 http://
↪schemas.opengis.net/wps/1.0.0/wpsAll.xsd">
  <ows:Identifier>JTS:convexHull</ows:Identifier>
  <wps:DataInputs>
    <wps:Input>
      <ows:Identifier>geom</ows:Identifier>
      <wps:Reference mimeType="application/wkt" xlink:href="http://www.geo-solutions.
↪it/geoserver/wfs?" method="GET"/>
    </wps:Input>
  </wps:DataInputs>
  <wps:ResponseForm>
    <wps:ResponseDocument lineage="true" storeExecuteResponse="true" status="true">
      <wps:Output asReference="false">
        <ows:Identifier>result</ows:Identifier>
      </wps:Output>
    </wps:ResponseDocument>
  </wps:ResponseForm>
</wps:Execute>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<wps:Execute version="1.0.0" service="WPS" xmlns:xsi="http://www.w3.org/2001/
↪XMLSchema-instance" xmlns="http://www.opengis.net/wps/1.0.0" xmlns:wfs="http://www.
↪opengis.net/wfs" xmlns:wps="http://www.opengis.net/wps/1.0.0" xmlns:ows="http://www.
↪opengis.net/ows/1.1" xmlns:gml="http://www.opengis.net/gml" xmlns:ogc="http://www.
↪opengis.net/ogc" xmlns:wcs="http://www.opengis.net/wcs/1.1.1" xmlns:xlink="http://
↪www.w3.org/1999/xlink" xsi:schemaLocation="http://www.opengis.net/wps/1.0.0 http://
↪schemas.opengis.net/wps/1.0.0/wpsAll.xsd">
```

```
  <ows:Identifier>gs:BufferFeatureCollection</ows:Identifier>
  <wps:DataInputs>
    <wps:Input>
      <ows:Identifier>features</ows:Identifier>
      <wps:Reference mimeType="text/xml" xlink:href="http://geoserver/wps" method=
↪"POST">
        <wps:Body>
            <wps:Execute version="1.0.0" service="WPS" xmlns:xsi="http://www.w3.org/
↪2001/XMLSchema-instance" xmlns="http://www.opengis.net/wps/1.0.0" xmlns:wfs="http://
↪www.opengis.net/wfs" xmlns:wps="http://www.opengis.net/wps/1.0.0" xmlns:ows="http://
↪www.opengis.net/ows/1.1" xmlns:gml="http://www.opengis.net/gml" xmlns:ogc="http://
↪www.opengis.net/ogc" xmlns:wcs="http://www.opengis.net/wcs/1.1.1" xmlns:xlink=
↪"http://www.w3.org/1999/xlink" xsi:schemaLocation="http://www.opengis.net/wps/1.0.0␣
↪http://schemas.opengis.net/wps/1.0.0/wpsAll.xsd">
              <ows:Identifier>gs:CollectGeometries</ows:Identifier>
              <wps:DataInputs>
                <wps:Input>
                  <ows:Identifier>features</ows:Identifier>
                  <wps:Reference mimeType="text/xml" xlink:href="http://geoserver/wfs
↪" method="POST">
                    <wps:Body>
                      <wfs:GetFeature service="WFS" version="1.0.0" outputFormat="GML2
↪" xmlns:geonode="http://www.geonode.org/">
                        <wfs:Query typeName="geonode:san_andres_y_providencia_
↪administrative"/>
                      </wfs:GetFeature>
                    </wps:Body>
                  </wps:Reference>
                </wps:Input>
              </wps:DataInputs>
              <wps:ResponseForm>
                <wps:RawDataOutput lineage="true" mimeType="text/xml; subtype=gml/3.1.
↪1">
                  <ows:Identifier>result</ows:Identifier>
                </wps:RawDataOutput>
              </wps:ResponseForm>
            </wps:Execute>
        </wps:Body>
      </wps:Reference>
    </wps:Input>
    <wps:Input>
      <ows:Identifier>distance</ows:Identifier>
      <wps:Data>
        <wps:LiteralData>0.005</wps:LiteralData>
      </wps:Data>
    </wps:Input>
  </wps:DataInputs>
  <wps:ResponseForm>
    <wps:ResponseDocument lineage="true" storeExecuteResponse="true" status="true">
      <wps:Output asReference="false">
        <ows:Identifier>result</ows:Identifier>
      </wps:Output>
    </wps:ResponseDocument>
  </wps:ResponseForm>
</wps:Execute>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<wps:Execute version="1.0.0" service="WPS" xmlns:xsi="http://www.w3.org/2001/
↪XMLSchema-instance" xmlns="http://www.opengis.net/wps/1.0.0" xmlns:wfs="http://www.
↪opengis.net/wfs" xmlns:wps="http://www.opengis.net/wps/1.0.0" xmlns:ows="http://www.
↪opengis.net/ows/1.1" xmlns:gml="http://www.opengis.net/gml" xmlns:ogc="http://www.
↪opengis.net/ogc" xmlns:wcs="http://www.opengis.net/wcs/1.1.1" xmlns:xlink="http://
↪www.w3.org/1999/xlink" xsi:schemaLocation="http://www.opengis.net/wps/1.0.0 http://
↪schemas.opengis.net/wps/1.0.0/wpsAll.xsd">
```

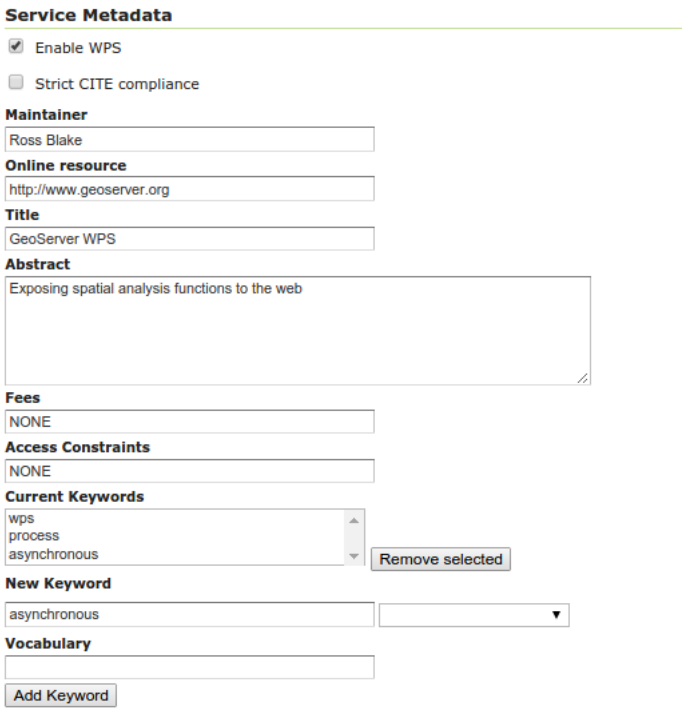
```
  <ows:Identifier>gs:Clip</ows:Identifier>
  <wps:DataInputs>
    <wps:Input>
      <ows:Identifier>features</ows:Identifier>
      <wps:Reference mimeType="text/xml" xlink:href="http://geoserver/wfs" method=
↪"POST">
        <wps:Body>
          <wfs:GetFeature service="WFS" version="1.0.0" outputFormat="GML2"␣
↪xmlns:geonode="http://www.geonode.org/">
            <wfs:Query typeName="geonode:san_andres_y_providencia_administrative"/>
          </wfs:GetFeature>
        </wps:Body>
      </wps:Reference>
    </wps:Input>
    <wps:Input>
      <ows:Identifier>clip</ows:Identifier>
      <wps:Data>
        <wps:ComplexData mimeType="application/json"><![CDATA[{"type":"MultiLineString
↪","coordinates":[[[-81.8254,12.199],[-81.8162,12.1827],[-81.812,12.1653],[-81.8156,
↪12.1465],[-81.8269,12.1321],[-81.8433,12.123],[-81.8614,12.119],[-81.8795,12.1232],
↪[-81.8953,12.1336],[-81.9049,12.1494],[-81.9087,12.1673],[-81.9054,12.1864],[-81.
↪8938,12.2004],[-81.8795,12.2089],[-81.8593,12.2136],[-81.8399,12.2096],[-81.8254,12.
↪199]],[[-81.6565,12.635],[-81.6808,12.6391],[-81.7085,12.6262],[-81.739,12.6046],[-
↪81.7611,12.5775],[-81.775,12.5397],[-81.7708,12.5207],[-81.7667,12.4971],[-81.7701,
↪12.4748],[-81.7646,12.4504],[-81.739,12.4369],[-81.7022,12.4389],[-81.6835,12.4578],
↪[-81.6794,12.4883],[-81.6676,12.5153],[-81.651,12.541],[-81.66,12.5552],[-81.6489,
↪12.5762],[-81.6274,12.5931],[-81.6309,12.6181],[-81.6565,12.635]],[[-81.2954,13.
↪3496],[-81.3004,13.3132],[-81.3143,13.29],[-81.3413,13.2755],[-81.3731,13.2674],[-
↪81.4058,13.2657],[-81.4335,13.2633],[-81.4531,13.2771],[-81.4574,13.3079],[-81.4663,
↪13.3257],[-81.463,13.3476],[-81.447,13.3674],[-81.4228,13.3879],[-81.412,13.4126],[-
↪81.403,13.4375],[-81.391,13.4582],[-81.3674,13.4687],[-81.3503,13.4574],[-81.3205,
↪13.448],[-81.2941,13.4177],[-81.2846,13.3878],[-81.2954,13.3496]],[[-79.9333,14.
↪9856],[-79.9333,15.5028]]]}]]></wps:ComplexData>
      </wps:Data>
    </wps:Input>
  </wps:DataInputs>
  <wps:ResponseForm>
    <wps:ResponseDocument lineage="true" storeExecuteResponse="true" status="true">
      <wps:Output asReference="false">
        <ows:Identifier>result</ows:Identifier>
      </wps:Output>
    </wps:ResponseDocument>
  </wps:ResponseForm>
</wps:Execute>
```

7.4.3 WPS Service page

The Web Processing Service (WPS) page supports the basic metadata for the service, as well as service specific settings

Service Metadata

The service metadata section is common among all services. See the section on *Service Metadata*.

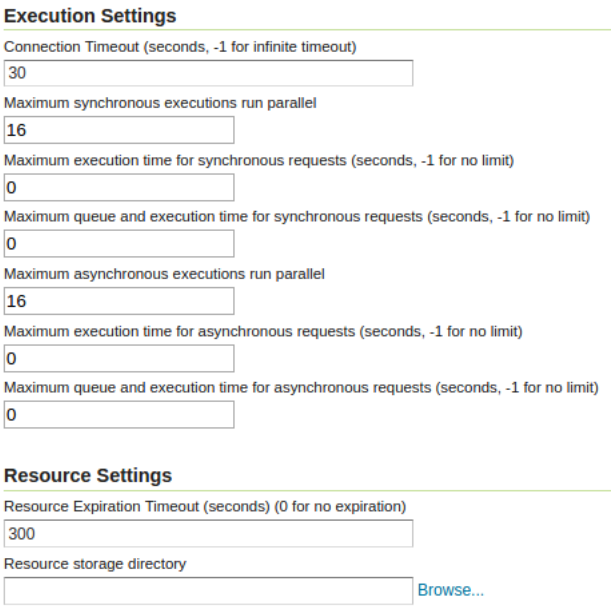


Execution and resource management options

Execution settings:

- *Connection timeout*: the number of seconds the WPS will wait before giving up on a remote HTTP connection used to retrieve complex inputs
- *Maximum synchronous executions run parallel*: the maximum number of synchronous processes that will run in parallel at a given time. The others will be queued.
- *Maximum execution time for synchronous requests*: the maximum time a synchronous process is allowed executing. Processes running in synchronous mode will have to complete execution within the set time limit, or they will be dismissed automatically. These requests have the client waiting for a response on a HTTP connection, so choose a relatively short time (e.g., 60 seconds)
- *Maximum queue and execution time for synchronous requests*: the maximum time a process is allowed in the queue and executing. Processes running in synchronous mode will have to complete within the set time limit, or they will be dismissed automatically. These requests have the client waiting for a response on a HTTP connection, so choose a relatively short time (e.g., 60 seconds)
- *Maximum asynchronous executions run parallel*: the maximum number of asynchronous processes that will run in parallel at a given time. The others will be queued
- *Maximum execution time for asynchronous requests*: the maximum time an asysynchronous process is allowed executing. Processes running in asynchronous mode will have to complete within the set time limit, or they will be dismissed automatically
- *Maximum queue and execution time for asynchronous requests*: the maximum time an asysynchronous process is allowed in the queue and executing. Processes running in asynchronous mode will have to complete within the set time limit, or they will be dismissed automatically

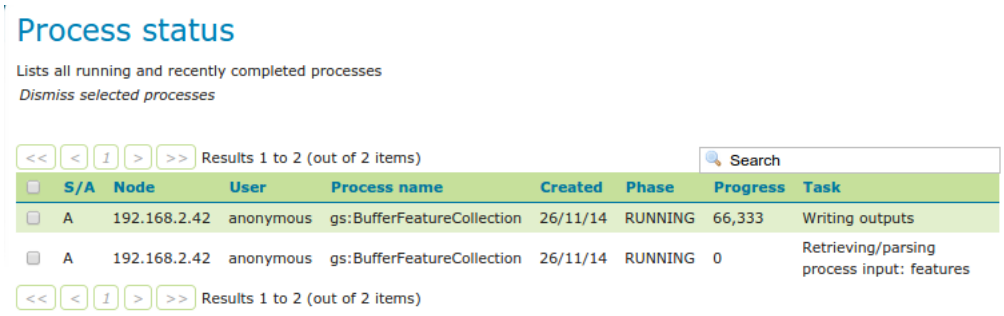
Resource settings:



- *Resource expiration timeout*: number of seconds the result of a asynchronous execution will be kept available on disk for user to retrieve. Once this time is expired these resources will be eligible for clearing (which happens at regular intervals).
- *Resource storage directory*: where on disk the input, temporary and output resources associated to a certain process will be kept. By default it will be the `temp/wps` directory inside the GeoServer data directory

Process status page

The process status page, available in the "About & Status" section, reports about running, and recently completed, processes:



The table contains several information bits:

- *S/A*: synchronous or asynchronous execution
- *Node*: the name of the machine running the process (important in a clustered environment)
- *User*: user that started the execution
- *Process name*: the main process being run (chained processes will not appear in this table)

- *Created*: when the process got created
- *Phase*: the current phase in the process lifecycle
- *Progress*: current progress
- *Task*: what the process is actually doing at the moment

In GeoServer there are the following execution phases:

- *QUEUED*: the process is waiting to be executed
- *RUNNING*: the process is either retriving and parsing the inputs, computing the results, or writing them out
- *FAILED*: the process execution terminated with a failure
- *SUCCESS*: the process execution terminated with a success
- *DISMISSING*: the process execution is being dismissed, depending on the process nature this might take some time, or be instantaneous

All executions listed in the table can be selected, and then dismissed using the "Dismiss selected processes" link at the top of the table. Unlike the "Dismiss" vendor operation this UI allows to also dismiss synchronous processes. Once the process dismissal is complete, the process execution will disappear from the table (in accordance with the WPS specification).

Completed processes can also be dismissed, this will cause all on disk resources associated to the processe to be removed immediately, instead of waiting for the regular time based expiration.

7.4.4 WPS Security and input limits

GeoServer service security is normally based on the generic *OGC security configuration*, however, when it comes to WPS there is also a need to **restrict access to individual processes**, in the same way that data security restricts access to layers.

WPS security allows access to be determined by process group or by single process. Each process and process group can be enabled/disabled, or subject to access control based on the user roles.

The WPS security configurations can be changed using the *Web administration interface* on the *WPS security* page under *Security*.

Setting access roles

The list of roles attached to each group or process will determine which users can access which processes. If the list is empty the group/process will be available to all users, unless it has been disabled, in which case it won't be available to anyone.

The roles string must be a list of roles separated by semicolons. The role editor provides auto-completion and also allows quick copy and paste of role lists from one process definition to the other:

Access modes

The process access mode configuration specifies how GeoServer will advertise secured processes and behave when a secured process is accessed without the necessary privileges. The parameter can be one of three values:

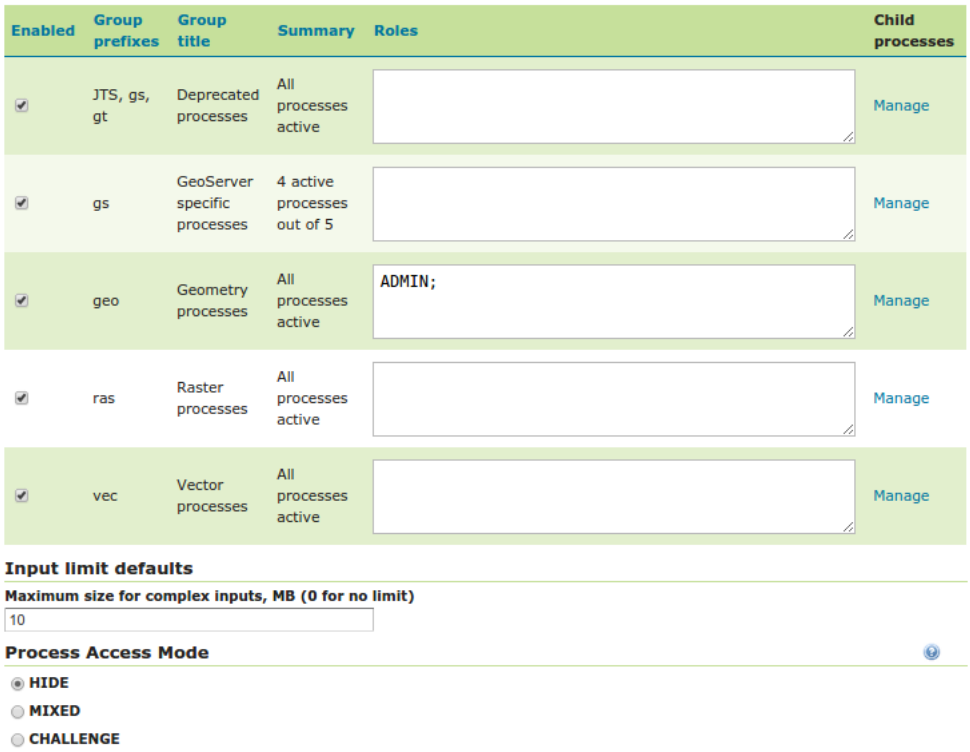


Fig. 7.69: The WPS security page

Fig. 7.70: Click to access the WPS security settings

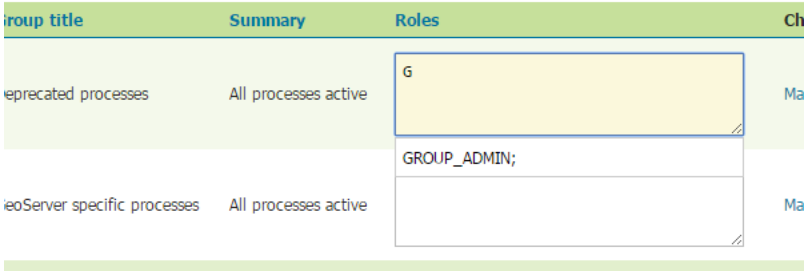


Fig. 7.71: Role selector field with auto-complete

- **HIDE** (default): The processes not available to the current user will be hidden from the user (not listed in the capabilities documents). Direct access will result in GeoServer claiming the process does not exist.
- **CHALLENGE**: All processes will be shown in the capabilities documents, but an authentication request will be raised if a secured process is specifically requested by a user that does not have sufficient access rights
- **MIXED**: The secured processes will not be shown in the capabilities documents for users not having sufficient access rights, but an authentication request will still be raised if a secured process is requested.

Input limits

The amount of resources used by a process is usually related directly to the inputs of the process itself. With this in mind, administrators can set three different type of limits on each process inputs:

- The maximum size of complex inputs
- The range of acceptable values for numeric values
- The maximum multiplicity of repeatable inputs

> **Note:** As an example of the last point, think of contour extraction, where the number of levels for the contours can drastically affect the execution time

GeoServer allows the administrator to configure these limits, and fail requests that don't respect them.

The maximum size can be given a global default on the *WPS security* page. It is also possible to define limits on a per-process basis by navigating to the process limits editor in the process list.

Note: Processes having a * beside the link have a defined set of limits

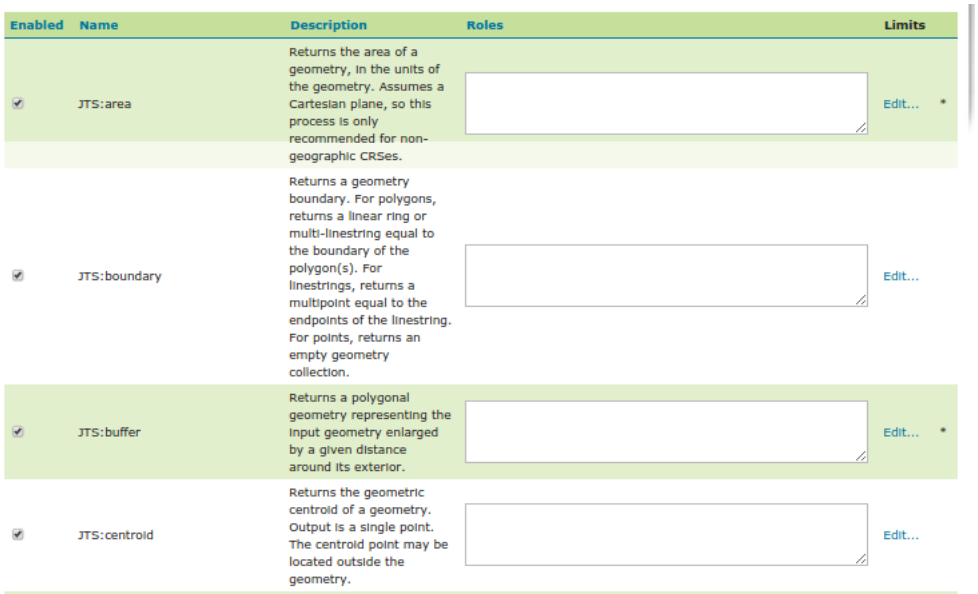


Fig. 7.72: The process selector, with access constraints and links to the limits configuration

The process limits editor shows all inputs for which a limit can be provided. An empty field means that limits are disabled for that input.

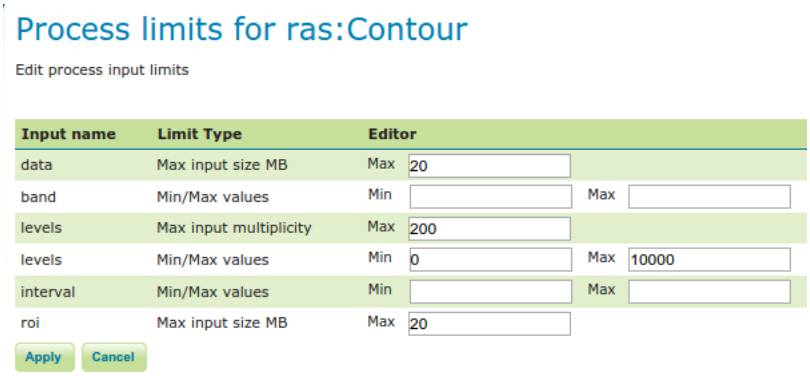


Fig. 7.73: The process limit page, with input limits configured

Warning: In order for the limits to be saved, click **both** *Apply* on this page and then *Submit* on the main WPS security page.

7.4.5 WPS Request Builder

The GeoServer WPS extension includes a request builder for testing out WPS processes through the *Web administration interface*. This tool can also be used to demonstrate processes, and construct your own examples.

Accessing the request builder

To access the WPS Request Builder:

1. Navigate to the main *Web administration interface*.
2. Click on the *Demos* link on the left side.
3. Select *WPS Request Builder* from the list of demos.

Fig. 7.74: *WPS request builder in the list of demos*

Using the request builder

The WPS Request Builder primarily consists of a selection box listing all of the available processes, and two buttons, one to submit the WPS request, and another to display what the POST request looks like.

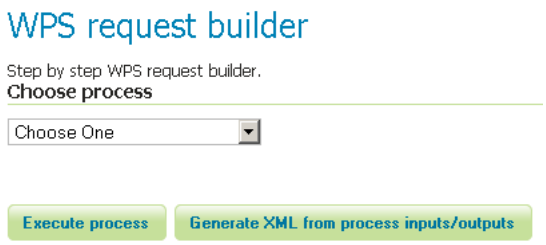


Fig. 7.75: *Blank WPS request builder form*

The display changes depending on the process and input selected. JTS processes have available as inputs any of a GML/WKT-based feature collection, URL reference, or subprocess. GeoServer-specific processes have all these as options and also includes the ability to choose a GeoServer layer as input.

For each process, a form will display based on the required and optional parameters associated with that process, if any.

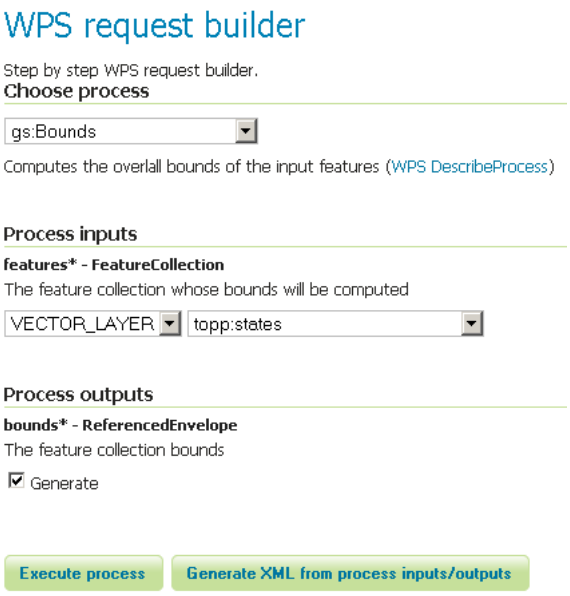


Fig. 7.76: *WPS request builder form to determine the bounds of topp:states*

To see the process as a POST request, click the *Generate XML from process inputs/outputs* button.

To execute the process, click the *Execute Process* button. The response will be displayed in a window or

7.4.6 Process Cookbook

The Web Processing Service describes a method for publishing geospatial processes, but does not specify what those processes should be. Servers that implement WPS therefore have complete leeway in what types of processes to implement, as well as how those processes are implemented. This means that a process request designed for one type of WPS is not expected to work on a different type of WPS.

GeoServer gathers processes into several different categories based on subject. These categories are grouped by prefix:

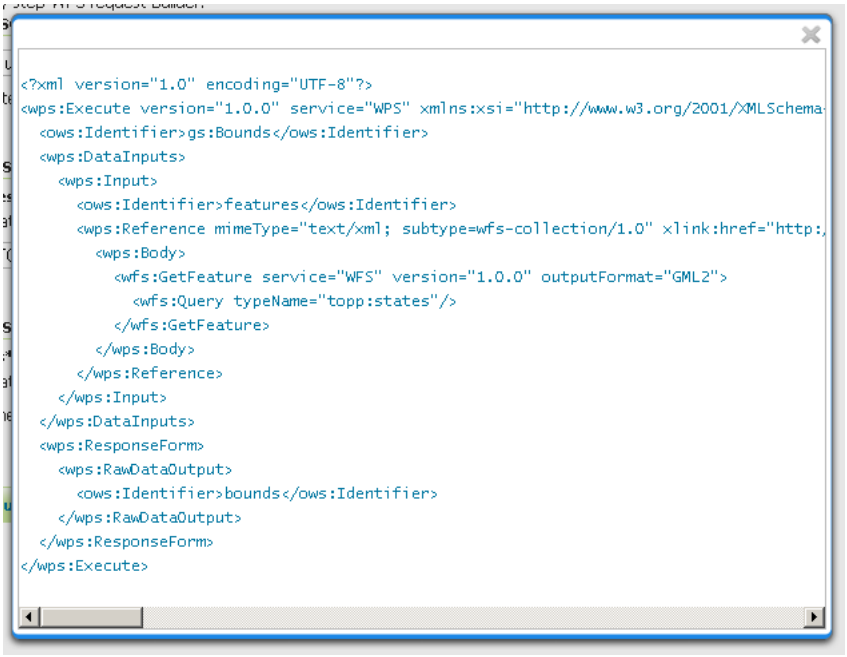


Fig. 7.77: *Raw WPS POST request for the above process*

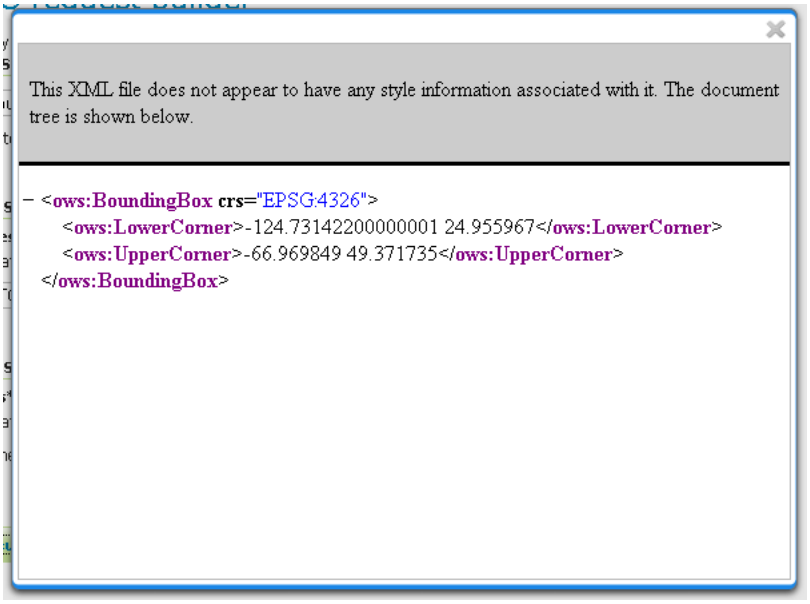


Fig. 7.78: *WPS server response*

- geo: geometry processes
- ras: raster processes
- vec: Vector processes
- gs: GeoServer-specific processes

This cookbook provides examples of some of the available process. Unless otherwise stated examples were generated with the *WPS Request Builder* using the sample data included with each GeoServer release.

Geometry Processes

The geometry processes are built using the JTS Topology Suite (JTS). JTS is a Java library of functions for processing geometries in two dimensions. JTS conforms to the Simple Features Specification for SQL published by the Open Geospatial Consortium (OGC), similar to PostGIS. JTS includes common spatial functions such as area, buffer, intersection, and simplify.

GeoServer WPS implements some of these functions as "geo" processes. The names and definitions of these processes are subject to change, so they have not been included here. For a full list of JTS processes, please see the GeoServer *WPS capabilities document* or browse with the *WPS Request Builder*.

GeoServer processes

GeoServer WPS includes a few processes created especially for use with GeoServer. These are usually GeoServer-specific functions, such as bounds and reprojection. They use an internal connection to the GeoServer WFS/WCS, not part of the WPS specification, for reading and writing data.

As with the "geo" processes, the names and definitions of these processes are subject to change, so they have not been included here. For a full list of GeoServer-specific processes, please see the GeoServer *WPS capabilities document* (or browse with the *WPS Request Builder*.)

Aggregation process

The aggregation process is used to perform common aggregation functions (sum, average, count) on vector data. The available outputs formats for this process are *text/xml* and *application/json*.

The process parameters are described in the table bellow:

Parameter	Description	Mandatory	Multiple
`features`	Input feature collection.	yes	no
`aggregationAttribute`	Attribute on which to perform aggregation.	yes	no
`function`	An aggregate function to compute. Functions include Count, Average, Max, Median, Min, StdDev, and Sum.	yes	yes
`singlePass`	If TRUE computes all aggregation values in a single pass. This will defeat DBMS-specific optimizations. If a group by attribute is provided this parameter will be ignored.	yes	no
`groupByAttributes`	Group by attribute.	no	yes

Follow some examples of the invocation of this process using GeoServer shipped *topp:states* layer.

The examples can be tested with CURL:

```
curl -u admin:geoserver -H 'Content-type: xml' -XPOST -d@'wps-request.xml' http://
↪localhost:8080/geoserver/wps
```

where *wps-request.xml* is the file that contains the request.

Aggregate Example

Counts the total number of states, sum all the number of persons, computes the average number of persons per state and give the maximum and minimum number of persons in a state.

Request:

```
<?xml version="1.0" encoding="UTF-8"?><wps:Execute version="1.0.0" service="WPS"␣
↪xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://www.opengis.net/
↪wps/1.0.0" xmlns:wfs="http://www.opengis.net/wfs" xmlns:wps="http://www.opengis.net/
↪wps/1.0.0" xmlns:ows="http://www.opengis.net/ows/1.1" xmlns:gml="http://www.opengis.
↪net/gml" xmlns:ogc="http://www.opengis.net/ogc" xmlns:wcs="http://www.opengis.net/
↪wcs/1.1.1" xmlns:xlink="http://www.w3.org/1999/xlink" xsi:schemaLocation="http://
↪www.opengis.net/wps/1.0.0 http://schemas.opengis.net/wps/1.0.0/wpsAll.xsd">
  <ows:Identifier>gs:Aggregate</ows:Identifier>
  <wps:DataInputs>
    <wps:Input>
      <ows:Identifier>features</ows:Identifier>
      <wps:Reference mimeType="text/xml" xlink:href="http://geoserver/wfs" method=
↪"POST">
        <wps:Body>
          <wfs:GetFeature service="WFS" version="1.0.0" outputFormat="GML2" xmlns:sf=
↪"http://www.openplans.org/spearfish">
            <wfs:Query typeName="topp:states"/>
          </wfs:GetFeature>
        </wps:Body>
      </wps:Reference>
    </wps:Input>
    <wps:Input>
      <ows:Identifier>aggregationAttribute</ows:Identifier>
      <wps:Data>
        <wps:LiteralData>PERSONS</wps:LiteralData>
      </wps:Data>
    </wps:Input>
    <wps:Input>
      <ows:Identifier>function</ows:Identifier>
      <wps:Data>
        <wps:LiteralData>Count</wps:LiteralData>
      </wps:Data>
    </wps:Input>
    <wps:Input>
      <ows:Identifier>function</ows:Identifier>
      <wps:Data>
        <wps:LiteralData>Average</wps:LiteralData>
      </wps:Data>
    </wps:Input>
    <wps:Input>
      <ows:Identifier>function</ows:Identifier>
      <wps:Data>
        <wps:LiteralData>Sum</wps:LiteralData>
      </wps:Data>
    </wps:Input>
    <wps:Input>
      <ows:Identifier>function</ows:Identifier>
      <wps:Data>
        <wps:LiteralData>Min</wps:LiteralData>
```

```
      </wps:Data>
    </wps:Input>
    <wps:Input>
      <ows:Identifier>function</ows:Identifier>
      <wps:Data>
        <wps:LiteralData>Max</wps:LiteralData>
      </wps:Data>
    </wps:Input>
    <wps:Input>
      <ows:Identifier>singlePass</ows:Identifier>
      <wps:Data>
        <wps:LiteralData>false</wps:LiteralData>
      </wps:Data>
    </wps:Input>
  </wps:DataInputs>
  <wps:ResponseForm>
    <wps:RawDataOutput mimeType="application/json">
      <ows:Identifier>result</ows:Identifier>
    </wps:RawDataOutput>
  </wps:ResponseForm>
</wps:Execute>
```

The result:

```
{
  "AggregationAttribute": "PERSONS",
  "AggregationFunctions": ["Max", "Min", "Average", "Sum", "Count"],
  "GroupByAttributes": [],
  "AggregationResults": [
    [29760021, 453588, 5038397.020408163, 246881454, 49]
  ]
}
```

The value of *AggregationResults* attribute should be read in a tabular way. The group by attributes come first in the order they appear in *GroupByAttributes* attribute. After comes the result of the aggregation functions in the order they appear in the *AggregationFunctions* attribute. In this case there is no group by attributes so the result only contains a row with the aggregation functions results. This is very similar to the result of an SQL query.

This result should be interpreted like this:

Max	Min	Average	Sum	Count
29760021	453588	5038397.020408163	246881454	49

To obtain the result in the XML format the request *wps:ResponseForm* element needs to be changed to:

```
<wps:ResponseForm>
  <wps:RawDataOutput mimeType="text/xml">
    <ows:Identifier>result</ows:Identifier>
  </wps:RawDataOutput>
</wps:ResponseForm>
```

The result in XML format:

```
<?xml version="1.0" encoding="UTF-8"?>
<AggregationResults>
  <Min>453588.0</Min>
```

```
  <Max>2.9760021E7</Max>
  <Average>5038397.020408163</Average>
  <Sum>2.46881454E8</Sum>
  <Count>49</Count>
</AggregationResults>
```

Aggregate GroupBy Example

This example count the number of states and the population average grouped by region.

Request:

```
<?xml version="1.0" encoding="UTF-8"?><wps:Execute version="1.0.0" service="WPS"␣
↪xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://www.opengis.net/
↪wps/1.0.0" xmlns:wfs="http://www.opengis.net/wfs" xmlns:wps="http://www.opengis.net/
↪wps/1.0.0" xmlns:ows="http://www.opengis.net/ows/1.1" xmlns:gml="http://www.opengis.
↪net/gml" xmlns:ogc="http://www.opengis.net/ogc" xmlns:wcs="http://www.opengis.net/
↪wcs/1.1.1" xmlns:xlink="http://www.w3.org/1999/xlink" xsi:schemaLocation="http://
↪www.opengis.net/wps/1.0.0 http://schemas.opengis.net/wps/1.0.0/wpsAll.xsd">
  <ows:Identifier>gs:Aggregate</ows:Identifier>
  <wps:DataInputs>
    <wps:Input>
      <ows:Identifier>features</ows:Identifier>
      <wps:Reference mimeType="text/xml" xlink:href="http://geoserver/wfs" method=
↪"POST">
        <wps:Body>
          <wfs:GetFeature service="WFS" version="1.0.0" outputFormat="GML2" xmlns:sf=
↪"http://www.openplans.org/spearfish">
            <wfs:Query typeName="topp:states"/>
          </wfs:GetFeature>
        </wps:Body>
      </wps:Reference>
    </wps:Input>
    <wps:Input>
      <ows:Identifier>aggregationAttribute</ows:Identifier>
      <wps:Data>
        <wps:LiteralData>PERSONS</wps:LiteralData>
      </wps:Data>
    </wps:Input>
    <wps:Input>
      <ows:Identifier>function</ows:Identifier>
      <wps:Data>
        <wps:LiteralData>Count</wps:LiteralData>
      </wps:Data>
    </wps:Input>
    <wps:Input>
      <ows:Identifier>function</ows:Identifier>
      <wps:Data>
        <wps:LiteralData>Average</wps:LiteralData>
      </wps:Data>
    </wps:Input>
    <wps:Input>
      <ows:Identifier>singlePass</ows:Identifier>
      <wps:Data>
        <wps:LiteralData>false</wps:LiteralData>
      </wps:Data>
    </wps:Input>
```

```
    <wps:Input>
      <ows:Identifier>groupByAttributes</ows:Identifier>
      <wps:Data>
        <wps:LiteralData>SUB_REGION</wps:LiteralData>
      </wps:Data>
    </wps:Input>
  </wps:DataInputs>
  <wps:ResponseForm>
    <wps:RawDataOutput mimeType="application/json">
      <ows:Identifier>result</ows:Identifier>
    </wps:RawDataOutput>
  </wps:ResponseForm>
</wps:Execute>
```

The result:

```
{
  "AggregationAttribute": "PERSONS",
  "AggregationFunctions": ["Average", "Count"],
  "GroupByAttributes": ["SUB_REGION"],
  "AggregationResults": [
    [ "N Eng", 2201157.1666666665, 6 ],
    [ "W N Cen", 2522812.8571428573, 7 ],
    [ "Pacific", 12489678, 3 ],
    [ "Mtn", 1690408.25, 8 ],
    [ "E S Cen", 3998821.25, 4 ],
    [ "S Atl", 4837695.666666667, 9 ],
    [ "Mid Atl", 12534095.333333334, 3 ],
    [ "E N Cen", 8209477.2, 5 ],
    [ "W S Cen", 6709575.75, 4 ]
  ]
}
```

Since there is a group by attribute the result contains a row for each different value of the group by attribute. Very similar to the result of an SQL query. If there is more that one group by attribute (which is not the case) their values will be in the order they appear in the *GroupByAttributes* attribute.

This result should be interpreted like this:

Sub Region	Average	count
N Eng	2201157.1666666665	6
W N Cen	2522812.8571428573	7
Pacific	12489678	3
Mtn	1690408.25	8
E S Cen	3998821.25	4
S Atl	4837695.666666667	9
Mid Atl	12534095.333333334	3
E N Cen	8209477.2	5
W S Cen	6709575.75	4

The result in XML format:

```
<?xml version="1.0" encoding="UTF-8"?>
<AggregationResults>
  <GroupByResult>
    <object-array>
```

```
      <string>N Eng</string>
      <double>2201157.1666666665</double>
      <int>6</int>
    </object-array>
    <object-array>
      <string>W N Cen</string>
      <double>2522812.8571428573</double>
      <int>7</int>
    </object-array>
    <object-array>
      <string>Pacific</string>
      <double>1.2489678E7</double>
      <int>3</int>
    </object-array>
    <object-array>
      <string>Mtn</string>
      <double>1690408.25</double>
      <int>8</int>
    </object-array>
    <object-array>
      <string>E S Cen</string>
      <double>3998821.25</double>
      <int>4</int>
    </object-array>
    <object-array>
      <string>S Atl</string>
      <double>4837695.666666667</double>
      <int>9</int>
    </object-array>
    <object-array>
      <string>Mid Atl</string>
      <double>1.2534095333333334E7</double>
      <int>3</int>
    </object-array>
    <object-array>
      <string>E N Cen</string>
      <double>8209477.2</double>
      <int>5</int>
    </object-array>
    <object-array>
      <string>W S Cen</string>
      <double>6709575.75</double>
      <int>4</int>
    </object-array>
  </GroupByResult>
</AggregationResults>
```

Process chaining

One of the benefits of WPS is its native ability to chain processes. Much like how functions can call other functions, a WPS process can use as its input the output of another process. Many complex functions can thus be combined in to a single powerful request.

For example, let's take some of the sample data that is shipped with GeoServer and use the WPS engine to chain a few of the built in processes, which will allow users to perform geospatial analysis on the fly.

The question we want to answer in this example is the following: How many miles of roads are crossing a

protected area?

The data that will be used for this example is included with a standard installation of GeoServer:

- *sf:roads*: the layer that contains road information
- *sf:restricted*: the layer representing restricted areas

The restricted areas partially overlap the roads. We would like to know the total length of roads inside the restricted areas, as shown in the next screenshot. The road network is represented in white against a false color DEM (Digital Elevation Model). The restricted areas are represented with a dashed line in dark brown. The portion of the road network that is inside the restricted areas is drawn in red.

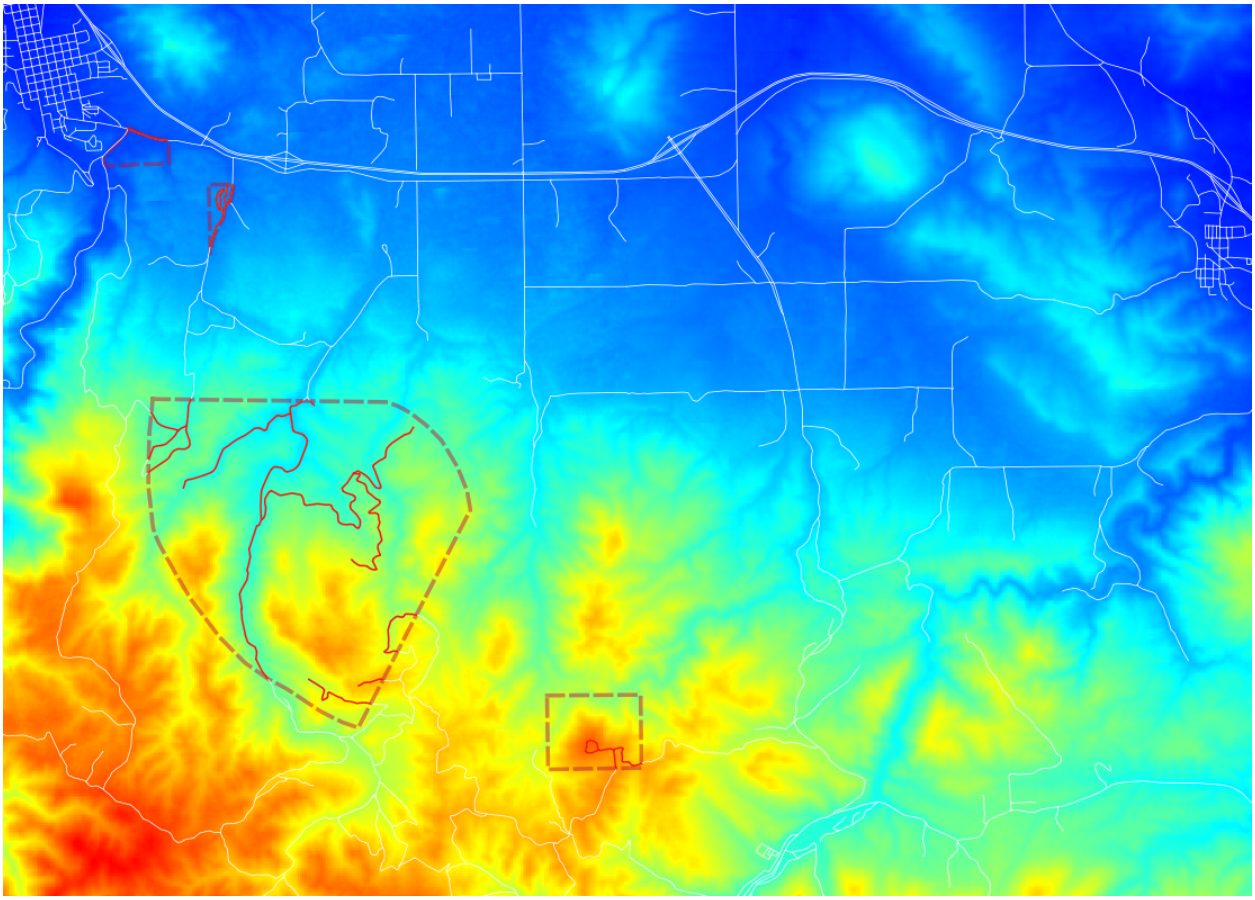


Fig. 7.79: Length of total roads inside restricted area

In order to calculate the total length, we will need the following built in WPS processes:

- `gs:IntersectionFeatureCollection`: returns the intersection between two feature collections adding the attributes from both of them
- `gs:CollectGeometries`: collects all the default geometries in a feature collection and returns them as a single geometry collection
- `JTS:length`: calculates the length of a geometry in the same unit of measure as the geometry

The sequence in which these processes are executed is important. The first thing we want to do is interesect the road network with the restricted areas. This gives us the feature collection with all the roads that we are interested in. Then we collect those geometries into a single GeometryCollection so that the length can be calculated with the built in JTS algorithm.

gs:IntersectionFeatureCollection –> gs:CollectGeometries –> JTS:length

The sequence of processes determines how the WPS request is built, by embedding the first process into the second, the second into the third, etc. A process produces some output which will become the input of the next process, resulting in a processing pipeline that can solve complex spatial analysis with a single HTTP request. The advantage of using GeoServer's layers is that data is not being shipped back and forth between processes, resulting in very good performance.

Here is the complete WPS request in XML format:

```
<?xml version="1.0" encoding="UTF-8"?>
<wps:Execute version="1.0.0" service="WPS" xmlns:xsi="http://www.w3.org/2001/
↪XMLSchema-instance" xmlns="http://www.opengis.net/wps/1.0.0" xmlns:wfs="http://www.
↪opengis.net/wfs" xmlns:wps="http://www.opengis.net/wps/1.0.0" xmlns:ows="http://www.
↪opengis.net/ows/1.1" xmlns:gml="http://www.opengis.net/gml" xmlns:ogc="http://www.
↪opengis.net/ogc" xmlns:wcs="http://www.opengis.net/wcs/1.1.1" xmlns:xlink="http://
↪www.w3.org/1999/xlink" xsi:schemaLocation="http://www.opengis.net/wps/1.0.0 http://
↪schemas.opengis.net/wps/1.0.0/wpsAll.xsd">

  <ows:Identifier>JTS:length</ows:Identifier>
  <wps:DataInputs>
    <wps:Input>
      <ows:Identifier>geom</ows:Identifier>
      <wps:Reference mimeType="text/xml; subtype=gml/3.1.1"
                     xlink:href="http://geoserver/wps" method="POST">
      <wps:Body>
        <wps:Execute version="1.0.0" service="WPS">
          <ows:Identifier>gs:CollectGeometries</ows:Identifier>
          <wps:DataInputs>
            <wps:Input>
              <ows:Identifier>features</ows:Identifier>
              <wps:Reference mimeType="text/xml; subtype=wfs-collection/1.0"␣
↪xlink:href="http://geoserver/wps" method="POST">
                <wps:Body>
                  <wps:Execute version="1.0.0" service="WPS">
                    <ows:Identifier>gs:IntersectionFeatureCollection</ows:Identifier>
                    <wps:DataInputs>
                      <wps:Input>
                        <ows:Identifier>first feature collection</ows:Identifier>
                        <wps:Reference mimeType="text/xml; subtype=wfs-collection/1.0
↪" xlink:href="http://geoserver/wfs" method="POST">
                          <wps:Body>
                            <wfs:GetFeature service="WFS" version="1.0.0"␣
↪outputFormat="GML2">
                              <wfs:Query typeName="sf:roads"/>
                            </wfs:GetFeature>
                          </wps:Body>
                        </wps:Reference>
                      </wps:Input>
                      <wps:Input>
                        <ows:Identifier>second feature collection</ows:Identifier>
                        <wps:Reference mimeType="text/xml; subtype=wfs-collection/1.0
↪" xlink:href="http://geoserver/wfs" method="POST">
                          <wps:Body>
                            <wfs:GetFeature service="WFS" version="1.0.0"␣
↪outputFormat="GML2">
                              <wfs:Query typeName="sf:restricted"/>
                            </wfs:GetFeature>
                          </wps:Body>
```

```
                        </wps:Reference>
                      </wps:Input>
                      <wps:Input>
                        <ows:Identifier>first attributes to retain</ows:Identifier>
                        <wps:Data>
                          <wps:LiteralData>the_geom cat</wps:LiteralData>
                        </wps:Data>
                      </wps:Input>
                      <wps:Input>
                        <ows:Identifier>second attributes to retain</ows:Identifier>
                        <wps:Data>
                          <wps:LiteralData>cat</wps:LiteralData>
                        </wps:Data>
                      </wps:Input>
                    </wps:DataInputs>
                    <wps:ResponseForm>
                      <wps:RawDataOutput mimeType="text/xml;
                                         subtype=wfs-collection/1.0">
                        <ows:Identifier>result</ows:Identifier>
                      </wps:RawDataOutput>
                    </wps:ResponseForm>
                  </wps:Execute>
                </wps:Body>
              </wps:Reference>
            </wps:Input>
          </wps:DataInputs>
          <wps:ResponseForm>
            <wps:RawDataOutput mimeType="text/xml; subtype=gml/3.1.1">
              <ows:Identifier>result</ows:Identifier>
            </wps:RawDataOutput>
          </wps:ResponseForm>
        </wps:Execute>
      </wps:Body>
    </wps:Reference>
    </wps:Input>
  </wps:DataInputs>
  <wps:ResponseForm>
    <wps:RawDataOutput>
      <ows:Identifier>result</ows:Identifier>
    </wps:RawDataOutput>
  </wps:ResponseForm>
</wps:Execute>
```

You can save this XML request in a file called wps-chaining.xml and execute the request using cURL like this:

curl -u admin:geoserver -H 'Content-type: xml' -XPOST -d@'wps-chaining.xml' http://localhost:8080/geoserver/wps

The response is just a number, the total length of the roads that intersect the restricted areas, and should be around *25076.285* meters (the length process returns map units)

To see WPS requests in action, you can use the built-in *WPS Request Builder*.

Note: Previous releases of GeoServer grouped processes not by subject, but by the internal library responsible for implementation. The "JTS" and "gt" prefixes can be enabled to preserve backwards compatibility, or you may safely disable them off - their functionality is correctly sorted into the "vec" and "geo" categories.

7.4.7 Hazelcast based process status clustering

Starting with version 2.7.0 GeoServer has a new WPS extension point allowing GeoServer nodes in the same cluster to share the status of current WPS requests. This is particularly important for asynchronous ones, as the client polling for the progress/results might not be hitting the same node that's currently running the requests.

The Hazelcast based status sharing module leverages the Hazelcast library to share the information about the current process status using a replicated map.

Installation

The installation of the module follows the usual process for most extensions:

- Stop GeoServer
- Unpack the contents of gs-wps-hazelcast-status.zip into the `geoserver/WEB-INF/lib` folder
- Restart GeoServer

Configuration

The module does not require any configuration in case the default behavior is suitable for the deploy environment.

By default, the module will use multicast messages to locate other nodes in the same cluster and will automatically start sharing information about the process status with them.

In case this is not satisfactory, a `hazelcast.xml` file can be created/edited in the root of the GeoServer data directory to modify the network connection methods.

The file is not using a GeoServer specific syntax, it's instead a regular Hazelcast configuration file with a simple distributed map declaration:

```
<hazelcast xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.hazelcast.com/schema/config
                              http://www.hazelcast.com/schema/config/hazelcast-
↪config-3.3.xsd"
  xmlns="http://www.hazelcast.com/schema/config">

  <!-- Protecting against accidental cluster joining -->
  <group>
    <name>geoserver</name>
    <password>geoserver</password>
  </group>

  <!--
     Make Hazelcast use log4j just like GeoServer. Remember to add:
       log4j.category.com.hazelcast=INFO
     in the geoserver logging configuration to see Hazelcast log messages
  -->
  <properties>
    <property name="hazelcast.logging.type">log4j</property>
  </properties>

  <!-- Network section, by default it enables multicast, tune it to use tcp in case
    multicast is not allowed, and list the nodes that make up a reasonable core of the
    cluster (e.g., machines that will never be all down at the same time) -->
```

```
  <network>
    <port auto-increment="true">5701</port>
    <join>
      <multicast enabled="true">
        <multicast-group>224.2.2.3</multicast-group>
        <multicast-port>54327</multicast-port>
      </multicast>
      <tcp-ip enabled="false">
        <interface>127.0.0.1</interface>
      </tcp-ip>
      <aws enabled="false">
        <access-key>my-access-key</access-key>
        <secret-key>my-secret-key</secret-key>
        <region>us-east-1</region>
      </aws>
    </join>
  </network>



  <!-- The WPS status map -->
  <map name="wpsExecutionStatusMap">
    <indexes>
      <!-- Add indexes to support the two most common queries -->
      <index ordered="false">executionId</index>
      <index ordered="true">completionTime</index>
    </indexes>
  </map>
</hazelcast>
```

In case a TCP based configuration is desired, one just needs to disable the multicast one, enable the tcp-ip one, and add a list of interface addresses in it that will form the core of the cluster. Not all nodes in the cluster need to be listed in said section, but a list long enough to ensure that not all the nodes in the list might go down at the same time: as long as at least one of said nodes lives, the cluster will maintain its integrity.

7.5 Catalog Services for the Web (CSW)

This section discusses the Catalog Services for Web (CSW) optional extension. With this extension, GeoServer supports retrieving and displaying items from the GeoServer catalog using the CSW service.

For more information on CSW, please refer to OGC OpenGIS Implementation Specification 07-006r1 and the OGC tutorial on CSW.

7.5.1 Installing Catalog Services for Web (CSW)

To install the CSW extension:

1. Visit the GeoServer Download and navigate to the download page for the version od GeoServer your are using. The **csw** download is listed under extensions. The file name is called `geoserver-*-csw-plugin.zip`, where `*` matches the version number of GeoServer you are using.
2. Extract this file and place the JARs in `WEB-INF/lib`.
3. Perform any configuration required by your servlet container, and then restart.

4. Verify that the module was installed correctly by going to the Welcome page of the *Web administration interface* and seeing that *CSW* is listed in the *Service Capabilities* list.

7.5.2 Catalog Services for the Web (CSW) features

Supported operations

The following standard CSW operations are currently supported:

- GetCapabilities
- GetRecords
- GetRecordById
- GetDomain
- DescribeRecord

(Starting with GeoServer 2.9.x, a new vendor operation has been added: *DirectDownload*)

The Internal Catalog Store supports filtering on both full x-paths as well as the "Queryables" specified in GetCapabilities.

Catalog stores

The default catalog store is the Internal Catalog Store, which retrieves information from the GeoServer's internal catalog. The Simple Catalog Store (`simple-store` module) adds an alternative simple store which reads the catalog data directly from files (mainly used for testing).

If there are multiple catalog stores present (for example, when the Simple Catalog Store module is loaded), set the Java system property `DefaultCatalogStore` to make sure that the correct catalog store will be used. To use the Internal Catalog Store, this property must be set to:

```
DefaultCatalogStore=org.geoserver.csw.store.internal.GeoServerInternalCatalogStore
```

To use the Simple Catalog Store:

```
DefaultCatalogStore=org.geoserver.csw.store.simple.GeoServerSimpleCatalogStore
```

Supported schemes

The Internal Catalog Store supports two metadata schemes:

- Dublin Core
- ISO Metadata Profile

Mapping Files

Mapping files are located in the `csw` directory inside the *GeoServer data directory*. Each mapping file must have the exact name of the record type name combined with the `.properties` extension. For example:

- Dublin Core mapping can be found in the file `csw/Record.properties` inside the data directory.
- ISO Metadata mapping can be found in the file `csw/MD_Metadata.properties` inside the data directory.

The mapping files take the syntax from Java properties files. The left side of the equals sign specifies the target field name or path in the metadata record, paths being separated with dots. The right side of the equals sign specifies any CQL expression that denotes the value of the target property. The CQL expression is applied to each ResourceInfo object in the catalog and can retrieve all properties from this object. These expressions can make use of literals, properties present in the ResourceInfo object, and all normal CQL operators and functions. There is also support for complex datastructures such as Maps using the dot notation and Lists using the bracket notation (Example mapping files are given below).

The properties in the ResourceInfo object that can be used are:

```
name
qualifiedName
nativeName
qualifiedNativeName
alias
title
abstract
description
metadata.?
namespace
namespace.prefix
namespace.name
namespace.uri
namespace.metadata.?
keywords
keywords[?]
keywords[?].value
keywords[?].language
keywords[?].vocabulary
keywordValues
keywordValues[?]
metadataLinks
metadataLinks[?]
metadataLinks[?].id
metadataLinks[?].about
metadataLinks[?].metadataType
metadataLinks[?].type
metadataLinks[?].content
latLonBoundingBox
latLonBoundingBox.dimension
latLonBoundingBox.lowerCorner
latLonBoundingBox.upperCorner
nativeBoundingBox
nativeBoundingBox.dimension
nativeBoundingBox.lowerCorner
nativeBoundingBox.upperCorner
srs
nativeCrs
projectionPolicy
enabled
advertised
catalog.defaultNamespace
catalog.defaultWorkspace
store.name
store.description
store.type
store.metadata.?
store.enabled
```

```
store.workspace
store.workspace.name
store.metadata.?
store.connectionParameters.?
store.error
```

Depending on whether the resource is a FeatureTypeInfo or a CoverageInfo, additional properties may be taken from their respective object structure. You may use *REST* to view an xml model of feature types and datastores in which the xml tags represent the available properties in the objects.

Some fields in the metadata schemes can have multiple occurences. They may be mapped to properties in the Catalog model that are also multi-valued, such as for example `keywords`. It is also possible to use a filter function called `list` to map multiple single-valued or multi-valued catalog properties to a MetaData field with multiple occurences (see in ISO MetaData Profile example, mapping for the `identificationInfo.AbstractMD_Identification.citation.CI_Citation.alternateTitle` field).

Placing the `@` symbol in front of the field will set that to use as identifier for each metadata record. This may be useful for ID filters. Use a `$` sign in front of fields that are required to make sure the mapping is aware of the requirement (specifically for the purpose of property selection).

Dublin Core

Below is an example of a Dublin Core mapping file:

```
@identifier.value=id
title.value=title
creator.value='GeoServer Catalog'
subject.value=keywords
subject.scheme='http://www.digest.org/2.1'
abstract.value=abstract
description.value=strConcat('description about ' , title)
date.value="metadata.date"
type.value='http://purl.org/dc/dcmitype/Dataset'
publisher.value='Niels Charlier'
#format.value=
#language.value=
#coverage.value=
#source.value=
#relation.value=
#rights.value=
#contributor.value=
```

All fields have the form of `<fieldname>.value` for the actual value in the field. Additionally `<fieldname>.scheme` can be specified for the `@scheme` attribute of this field.

Examples of attributes extracted from the ResourceInfo are `id`, `title`, and `keywords`, etc. The attribute `metadata.date` uses the metadata (`java.util.`)Map from the Resource object. In this map, it searches for the keyword "date".

Note that double quotes are necessary in order to preserve this meaning of the dots.

ISO Metadata Profile

Below is an example of an ISO Metadata Profile Mapping File:

```
@fileIdentifier.CharacterString=id
identificationInfo.AbstractMD_Identification.citation.CI_Citation.title.
↪CharacterString=title
identificationInfo.AbstractMD_Identification.citation.CI_Citation.alternateTitle.
↪CharacterString=list(description,alias,strConcat('##',title))
identificationInfo.AbstractMD_Identification.descriptiveKeywords.MD_Keywords.keyword.
↪CharacterString=keywords
identificationInfo.AbstractMD_Identification.abstract.CharacterString=abstract
$dateStamp.Date= if_then_else ( isNull("metadata.date") , 'Unknown', "metadata.date")
hierarchyLevel.MD_ScopeCode.@codeListValue='http://purl.org/dc/dcmitype/Dataset'
$contact.CI_ResponsibleParty.individualName.CharacterString=
```

The full path of each field must be specified (separated with dots). XML attributes are specified with the `@` symbol, similar to the usual XML X-path notation.

To keep the result XSD compliant, the parameters `dateStamp.Date` and `contact.CI_ResponsibleParty.individualName.CharacterString` must be preceded by a `$` sign to make sure that they are always included even when using property selection.

For more information on the ISO Metadata standard, please see the OGC Implementation Specification 07-045.

7.5.3 DirectDownload

DirectDownload is a new operation (since GeoServer 2.9.x) supported by the CSW service.

In the Meteorology, Oceanography and Earth Observation domains, layers are usually based on complex NetCDF/GRIB files. Protocols such as WCS are set up to allow slice, rescale and reprojection of data, but not to preserve the original data as is.

This new operation allows direct download of the raw data for that layer. If the DirectDownload capability is enabled for a Coverage layer, the CSW record will be updated to contain links pointing back to the CSW service, using the DirectDownload vendor operation that will assemble the files for the requested resource, zip them, and send the result back to the requester.

The download links (one for each data file composing the layer plus a link to get all the files composing the layer) are added as new entries in CSW records:

- as additional **term-references** element for a Dublin Core schema
- as additional **OnlineResource** for ISO Metadata

Those links also contain the validity domain for the file such as envelope/time/elevation/custom dimensions (when present) for multidimensional layers.

Configuration

DirectDownload capability can be activated as default for all layers, as global CSW configuration. Go into the CSW service panel and click on the *enable DirectDownload* checkbox if you want it enabled for all layers:

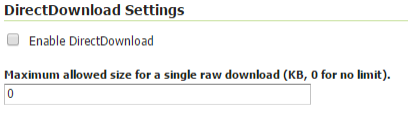


Fig. 7.80: *DirectDownload configuration (Service level)*

From this section you can also set a download size limit value (0 means no limit). The specified value represents the maximum size (in kilobytes) of the sum of the sizes of the raw data referred to by a single download link. (You can think about the case of a download link referring to the whole layer data which may contain a wide set of files).

Note that the size check is performed on the raw data files prior to any compression.

Per Layer configuration

DirectDownload capability can also be enabled/disabled for a specific layer, which will override the global CSW configuration.

Go to the *publishing* tab of the layer.

Fig. 7.81: *Layer publishing section*

Look for the *DirectDownload settings* section.

Fig. 7.82: *DirectDownload configuration (Layer level)*

The configuration of this parameter follows the same rules as shown for the CSW configuration panel.

GetRecords example

A GetRecords response containing a layer with DirectDownload enabled, may result having a piece like this (using ISO Metadata output schema):

```
...
<gmd:CI_OnlineResource>
  <gmd:linkage>
    <gmd:URL>
    http://localhost:8080/geoserver/ows?service=CSW&version=2.0.2&
↪request=DirectDownload&resourceId=geosolutions:Reflectivity_height_above_ground&
↪file=82643c5bf682f67ef8b7de737b90ada759965cd8-samplefile.grib2&ENVELOPE=-2699073.
↪2421875,-1588806.0302734375,2697926.7578125,1588193.9697265625&TIME=2015-06-
↪23T00:00:00.000Z/2015-06-23T00:00:00.000Z&HEIGHT_ABOVE_GROUND=1000.0/4000.0
    </gmd:URL>
  </gmd:linkage>
</gmd:CI_OnlineResource>
...
```

That URL allows the direct download of the indicated file. Note that the filename has a SHA-1 header to avoid publishing the underlying file system structure paths.

7.5.4 Catalog Services for the Web (CSW) tutorial

This tutorial will show how to use the CSW module. It assumes a fresh installation of GeoServer with the *CSW module installed*.

Configuration

In the `<data_dir>/csw` directory, create a new file named `MD_Metadata` (ISO Metadata Profile mapping file) with the following contents:

```
@fileIdentifier.CharacterString=prefixedName
identificationInfo.AbstractMD_Identification.citation.CI_Citation.title.
↪CharacterString=title
identificationInfo.AbstractMD_Identification.descriptiveKeywords.MD_Keywords.keyword.
↪CharacterString=keywords
identificationInfo.AbstractMD_Identification.abstract.CharacterString=abstract
$dateStamp.Date= if_then_else ( isNull("metadata.date") , 'Unknown', "metadata.date")
hierarchyLevel.MD_ScopeCode.@codeListValue='http://purl.org/dc/dcmitype/Dataset'
$contact.CI_ResponsibleParty.individualName.CharacterString='John Smith'
```

Services

With GeoServer running (and responding on `http://localhost:8080`), test GeoServer CSW in a web browser by querying the CSW capabilities as follows:

```
http://localhost:8080/geoserver/csw?service=csw&version=2.0.2&request=GetCapabilities
```

We can request a description of our Metadata record:

```
http://localhost:8080/geoserver/csw?service=CSW&version=2.0.2&request=DescribeRecord&
↪typeName=gmd:MD_Metadata
```

This yields the following result:

```
<?xml version="1.0" encoding="UTF-8"?>
<csw:DescribeRecordResponse xmlns:csw="http://www.opengis.net/cat/csw/2.0.2"␣
↪xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://
↪www.opengis.net/cat/csw/2.0.2 http://localhost:8080/geoserver/schemas/csw/2.0.2CSW-
↪discovery.xsd">
<csw:SchemaComponent targetNamespace="http://www.opengis.net/cat/csw/2.0.2"␣
↪schemaLanguage="http://www.w3.org/XML/Schema">
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xlink="http://www.w3.org/
↪1999/xlink" xmlns:gco="http://www.isotc211.org/2005/gco" xmlns:gmd="http://www.
↪isotc211.org/2005/gmd" targetNamespace="http://www.isotc211.org/2005/gmd"␣
↪elementFormDefault="qualified" version="2012-07-13">
      <!-- ================================== Annotation␣
↪================================== -->
      <xs:annotation>
              <xs:documentation>Geographic MetaData (GMD) extensible markup language␣
↪is a component of the XML Schema Implementation of Geographic Information Metadata␣
↪documented in ISO/TS 19139:2007. GMD includes all the definitions of http://www.
↪isotc211.org/2005/gmd namespace. The root document of this namespace is the file␣
↪gmd.xsd. This identification.xsd schema implements the UML conceptual schema␣
↪defined in A.2.2 of ISO 19115:2003. It contains the implementation of the following␣
↪classes: MD_Identification, MD_BrowseGraphic, MD_DataIdentification, MD_
↪ServiceIdentification, MD_RepresentativeFraction, MD_Usage, MD_Keywords, DS_
↪Association, MD_AggregateInformation, MD_CharacterSetCode, MD_
↪SpatialRepresentationTypeCode, MD_TopicCategoryCode, MD_ProgressCode, MD_
↪KeywordTypeCode, DS_AssociationTypeCode, DS_InitiativeTypeCode, MD_CoverageContentType</
↪xs:documentation>
```

```
      </xs:annotation>
      ...
```

Query all layers as follows:

```
http://localhost:8080/geoserver/csw?service=CSW&version=2.0.2&request=GetRecords&
↪typeNames=gmd:MD_Metadata&resultType=results&elementSetName=full&outputSchema=http:/
↪/www.isotc211.org/2005/gmd
```

Request a particular layer by ID...:

```
http://localhost:8080/geoserver/csw?service=CSW&version=2.0.2&request=GetRecordById&
↪elementsetname=summary&id=CoverageInfoImpl--4a9eec43:132d48aac79:-8000&
↪typeNames=gmd:MD_Metadata&resultType=results&elementSetName=full&outputSchema=http:/
↪/www.isotc211.org/2005/gmd
```

...or use a filter to retrieve it by Title:

```
http://localhost:8080/geoserver/csw?service=CSW&version=2.0.2&request=GetRecords&
↪typeNames=gmd:MD_Metadata&resultType=results&elementSetName=full&outputSchema=http:/
↪/www.isotc211.org/2005/gmd&constraint=Title=%27mosaic%27
```

Either case should return:

```
<?xml version="1.0" encoding="UTF-8"?>
<csw:GetRecordsResponse xmlns:xml="http://www.w3.org/XML/1998/namespace" xmlns="http:/
↪/www.opengis.net/cat/csw/apiso/1.0" xmlns:csw="http://www.opengis.net/cat/csw/2.0.2
↪" xmlns:gco="http://www.isotc211.org/2005/gco" xmlns:gmd="http://www.isotc211.org/
↪2005/gmd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="2.0.2"␣
↪xsi:schemaLocation="http://www.opengis.net/cat/csw/2.0.2 http://localhost:8080/
↪geoserver/schemas/csw/2.0.2/record.xsd">
  <csw:SearchStatus timestamp="2013-06-28T13:41:43.090Z"/>
  <csw:SearchResults numberOfRecordsMatched="1" numberOfRecordsReturned="1"␣
↪nextRecord="0" recordSchema="http://www.isotc211.org/2005/gmd" elementSet="full">
    <gmd:MD_Metadata>
      <gmd:fileIdentifier>
        <gco:CharacterString>CoverageInfoImpl--4a9eec43:132d48aac79:-8000</
↪gco:CharacterString>
      </gmd:fileIdentifier>
      <gmd:dateStamp>
        <gco:Date>Unknown</gco:Date>
      </gmd:dateStamp>
      <gmd:identificationInfo>
        <gmd:MD_DataIdentification>
          <gmd:extent>
            <gmd:EX_Extent>
              <gmd:geographicElement>
                <gmd:EX_GeographicBoundingBox crs="urn:x-ogc:def:crs:EPSG:6.11:4326">
                  <gmd:westBoundLongitude>36.492</gmd:westBoundLongitude>
                  <gmd:southBoundLatitude>6.346</gmd:southBoundLatitude>
                  <gmd:eastBoundLongitude>46.591</gmd:eastBoundLongitude>
                  <gmd:northBoundLatitude>20.83</gmd:northBoundLatitude>
                </gmd:EX_GeographicBoundingBox>
              </gmd:geographicElement>
            </gmd:EX_Extent>
          </gmd:extent>
        </gmd:MD_DataIdentification>
        <gmd:AbstractMD_Identification>
```

```
          <gmd:citation>
            <gmd:CI_Citation>
              <gmd:title>
                <gco:CharacterString>mosaic</gco:CharacterString>
              </gmd:title>
            </gmd:CI_Citation>
          </gmd:citation>
          <gmd:descriptiveKeywords>
            <gmd:MD_Keywords>
              <gmd:keyword>
                <gco:CharacterString>WCS</gco:CharacterString>
              </gmd:keyword>
              <gmd:keyword>
                <gco:CharacterString>ImageMosaic</gco:CharacterString>
              </gmd:keyword>
              <gmd:keyword>
                <gco:CharacterString>mosaic</gco:CharacterString>
              </gmd:keyword>
            </gmd:MD_Keywords>
          </gmd:descriptiveKeywords>
        </gmd:AbstractMD_Identification>
      </gmd:identificationInfo>
      <gmd:contact>
        <gmd:CI_ResponsibleParty>
          <gmd:individualName>
            <gco:CharacterString>John Smith</gco:CharacterString>
          </gmd:individualName>
        </gmd:CI_ResponsibleParty>
      </gmd:contact>
      <gmd:hierarchyLevel>
        <gmd:MD_ScopeCode codeListValue="http://purl.org/dc/dcmitype/Dataset"/>
      </gmd:hierarchyLevel>
    </gmd:MD_Metadata>
  </csw:SearchResults>
</csw:GetRecordsResponse>
```

We can request the domain of a property. For example, all values of "Title":

```
http://localhost:8080/geoserver/csw?service=csw&version=2.0.2&request=GetDomain&
↪propertyName=Title
```

This should yield the following result:

```
<?xml version="1.0" encoding="UTF-8"?>
<csw:GetDomainResponse xmlns:csw="http://www.opengis.net/cat/csw/2.0.2" xmlns:dc=
↪"http://purl.org/dc/elements/1.1/" xmlns:dct="http://purl.org/dc/terms/" xmlns:ows=
↪"http://www.opengis.net/ows/1.1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
↪instance" xsi:schemaLocation="http://www.opengis.net/cat/csw/2.0.2 http://
↪localhost:8080/geoserver/schemas/csw/2.0.2/CSW-discovery.xsd">
  <csw:DomainValues type="csw:Record">
      <csw:PropertyName>Title</csw:PropertyName>
      <csw:ListOfValues>
        <csw:Value>A sample ArcGrid file</csw:Value>
        <csw:Value>Manhattan (NY) landmarks</csw:Value>
        <csw:Value>Manhattan (NY) points of interest</csw:Value>
        <csw:Value>Manhattan (NY) roads</csw:Value>
        <csw:Value>North America sample imagery</csw:Value>
        <csw:Value>Pk50095 is a A raster file accompanied by a spatial data file</
↪csw:Value>
```

```
        <csw:Value>Spearfish archeological sites</csw:Value>
        <csw:Value>Spearfish bug locations</csw:Value>
        <csw:Value>Spearfish restricted areas</csw:Value>
        <csw:Value>Spearfish roads</csw:Value>
        <csw:Value>Spearfish streams</csw:Value>
        <csw:Value>Tasmania cities</csw:Value>
        <csw:Value>Tasmania roads</csw:Value>
        <csw:Value>Tasmania state boundaries</csw:Value>
        <csw:Value>Tasmania water bodies</csw:Value>
        <csw:Value>USA Population</csw:Value>
        <csw:Value>World rectangle</csw:Value>
        <csw:Value>mosaic</csw:Value>
        <csw:Value>sfdem is a Tagged Image File Format with Geographic information</
↪csw:Value>
      </csw:ListOfValues>
  </csw:DomainValues>
</csw:GetDomainResponse>
```

CHAPTER 8

Filtering

Filtering allows selecting features that satisfy a specific set of conditions. Filters can be used in several contexts in GeoServer:

- in WMS requests, to select which features should be displayed on a map
- in WFS requests, to specify the features to be returned
- in SLD documents, to apply different symbolization to features on a thematic map.

8.1 Supported filter languages

Data filtering in GeoServer is based on the concepts found in the OGC Filter Encoding Specification.

GeoServer accepts filters encoded in two different languages: *Filter Encoding* and *Common Query Language*.

8.1.1 Filter Encoding

The **Filter Encoding** language is an XML-based method for defining filters. XML Filters can be used in the following places in GeoServer:

- in WMS `GetMap` requests, using the `filter` parameter
- in WFS `GetFeature` requests, using the `filter` parameter
- in SLD Rules, in the *Filter* element

The Filter Encoding language is defined by OGC Filter Encoding Standards:

- Filter Encoding 1.0 is used in WFS 1.0 and SLD 1.0
- Filter Encoding 1.1 is used in WFS 1.1
- Filter Encoding 2.0 is used in WFS 2.0

8.1.2 CQL/ECQL

CQL (Common Query Language) is a plain-text language created for the *OGC Catalog* specification. GeoServer has adapted it to be an easy-to-use filtering mechanism. GeoServer actually implements a more powerful extension called **ECQL (Extended CQL)**, which allows expressing the full range of filters that *OGC Filter 1.1* can encode. ECQL is accepted in many places in GeoServer:

- in WMS `GetMap` requests, using the *cql_filter* parameter
- in WFS `GetFeature` requests, using the *cql_filter* parameter
- in SLD *dynamic symbolizers*

The *ECQL Reference* describes the features of the ECQL language. The *CQL and ECQL* tutorial shows examples of defining filters.

The CQL and ECQL languages are defined in:

- OpenGIS Catalog Services Specification contains the standard definition of CQL
- ECQL Grammar is the grammar defining the GeoTools ECQL implementation

8.2 Filter Encoding Reference

This is a reference for the **Filter Encoding** language implemented in GeoServer. The Filter Encoding language uses an XML-based syntax. It is defined by the OGC Filter Encoding standard.

Filters are used to select features or other objects from the context in which they are evaluated. They are similar in functionality to the SQL "WHERE" clause. A filter is specified using a **condition**.

8.2.1 Condition

A condition is a single *Predicate* element, or a combination of conditions by *Logical operators*.

8.2.2 Predicate

Predicates are boolean-valued expressions which compute relationships between values. A predicate is specified by using a **comparison operator** or a **spatial operator**. The operators are used to compare properties of the features being filtered to other feature properties or to literal data.

Comparison operators

Comparison operators are used to specify conditions on non-spatial attributes.

Binary Comparison operators

The **binary comparison operators** are:

- `<PropertyIsEqualTo>`
- `<PropertyIsNotEqualTo>`
- `<PropertyIsLessThan>`
- `<PropertyIsLessThanOrEqualTo>`

- `<PropertyIsGreaterThan>`
- `<PropertyIsGreaterThanOrEqualTo>`

They contain the elements:

Element	Required?	Description
Expression	Yes	The first value to compare. Often a `<PropertyName>`.
Expression	Yes	The second value to compare

Binary comparison operator elements may include an optional `matchCase` attribute, with the value `true` or `false`. If this attribute is `true` (the default), string comparisons are case-sensitive. If the attribute is `false` strings comparisons do not check case.

PropertyIsLike operator

The `<PropertyIsLike>` operator matches a string property value against a text **pattern**. It contains the elements:

Element	Required?	Description
`<PropertyName>`	Yes	Contains a string specifying the name of the property to test
`<Literal>`	Yes	Contains a pattern string to be matched

The pattern is specified by a sequence of regular characters and three special pattern characters. The pattern characters are defined by the following *required* attributes of the `<PropertyIsLike>` element:

- `wildCard` specifies the pattern character which matches any sequence of zero or more string characters
- `singleChar` specifies the pattern character which matches any single string character
- `escapeChar` specifies the escape character which can be used to escape the pattern characters

PropertyIsNull operator

The `<PropertyIsNull>` operator tests whether a property value is null. It contains the element:

Element	Required?	Description
`<PropertyName>`	Yes	contains a string specifying the name of the property to be tested

PropertyIsBetweeen operator

The `<PropertyIsBetween>` operator tests whether an expression value lies within a range given by a lower and upper bound (inclusive). It contains the elements:

Element	Required?	Description
Expression	Yes	The value to test
`<LowerBoundary>`	Yes	Contains an *Expression* giving the lower bound of the range
`<UpperBoundary>`	Yes	Contains an *Expression* giving the upper bound of the range

Spatial operators

Spatial operators are used to specify conditions on the geometric attributes of a feature. The following spatial operators are available:

Topological operators

These operators test topological spatial relationships using the standard OGC Simple Features predicates:

- `<Intersects>` - Tests whether two geometries intersect
- `<Disjoint>` - Tests whether two geometries are disjoint
- `<Contains>` - Tests whether a geometry contains another one
- `<Within>` - Tests whether a geometry is within another one
- `<Touches>` - Tests whether two geometries touch
- `<Crosses>` - Tests whether two geometries cross
- `<Overlaps>` - Tests whether two geometries overlap
- `<Equals>` - Tests whether two geometries are topologically equal

These contains the elements:

Element	Required?	Description
`<PropertyName>`	Yes	Contains a string specifying the name of the geometry-valued property to be tested.
GML Geometry	Yes	A GML literal value specifying the geometry to test against

Distance operators

These operators test distance relationships between a geometry property and a geometry literal:

- `<DWithin>`
- `<Beyond>`

They contain the elements:

Element	Required?	Description
`<PropertyName>`	Yes	Contains a string specifying the name of the property to be tested. If omitted, the *default geometry attribute* is assumed.
GML Geometry	Yes	A literal value specifying a geometry to compute the distance to. This may be either a geometry or an envelope in GML 3 format
`<Distance>`	Yes	Contains the numeric value for the distance tolerance. The element may include an optional `units` attribute.

Bounding Box operator

The `<BBOX>` operator tests whether a geometry-valued property intersects a fixed bounding box. It contains the elements:

Element	Required?	Description
<PropertyName>	No	Contains a string specifying the name of the property to be tested. If omitted, the *default geometry attribute* is assumed.
<gml:Box>	Yes	A GML Box literal value specifying the bounding box to test against

Examples

- This filter selects features with a geometry that intersects the point (1,1).

```
<Intersects>
  <PropertyName>GEOMETRY</PropertyName>
  <gml:Point>
    <gml:coordinates>1 1</gml:coordinates>
  </gml:Point>
</Intersects>
```

- This filter selects features with a geometry that overlaps a polygon.

```
<Overlaps>
  <PropertyName>Geometry</PropertyName>
  <gml:Polygon srsName="http://www.opengis.net/gml/srs/epsg.xml#63266405">
    <gml:outerBoundaryIs>
      <gml:LinearRing>
         <gml:posList> ... </gml:posList>
      </gml:LinearRing>
    </gml:outerBoundaryIs>
  </gml:Polygon>
 </Overlaps>
```

- This filter selects features with a geometry that intersects the geographic extent [-10,0 : 10,10].

```
<BBOX>
  <PropertyName>GEOMETRY</PropertyName>
  <gml:Box srsName="urn:x-ogc:def:crs:EPSG:4326">
    <gml:coord>
      <gml:X>-10</gml:X> <gml:Y>0</gml:Y>
    </gml:coord>
    <gml:coord>
      <gml:X>10</gml:X> <gml:Y>10</gml:Y>
    </gml:coord>
  </gml:Box>
</BBOX>
```

8.2.3 Logical operators

Logical operators are used to specify logical combinations of *Condition* elements (which may be either *Predicate* elements or other **logical operators**). They may be nested to any depth.

The following logical operators are available:

- <And> - computes the logical conjunction of the operands
- <Or> - computes the logical disjunction of the operands

The content for <And> and <Or> is two operands given by *Condition* elements.

- <Not> - computes the logical negation of the operand

The content for <Not> is a single operand given by a *Condition* element.

Examples

- This filter uses <And> to combine a comparison predicate and a spatial predicate:

```
<And>
   <PropertyIsEqualTo>
      <PropertyName>NAME</PropertyName>
      <Literal>New York</Literal>
   </PropertyIsEqualTo>
   <Intersects>
      <PropertyName>GEOMETRY</PropertyName>
      <Literal>
         <gml:Point>
             <gml:coordinates>1 1</gml:coordinates>
         </gml:Point>
      </Literal>
   </Intersects>
</And>
```

8.2.4 Expression

Filter expressions specify constant, variable or computed data values. An expression is formed from one of the following elements (some of which contain sub-expressions, meaning that expressions may be of arbitrary depth):

Arithmetic operators

The **arithmetic operator** elements compute arithmetic operations on numeric values.

- <Add> - adds the two operands
- <Sub> - subtracts the second operand from the first
- <Mul> - multiplies the two operands
- <Div> - divides the first operand by the second

Each arithmetic operator element contains two *Expression* elements providing the operands.

Function

The <Function> element specifies a filter function to be evaluated. The required name attribute gives the function name. The element contains a sequence of zero or more *Expression* elements providing the values of the function arguments.

See the *Filter Function Reference* for details of the functions provided by GeoServer.

Property Value

The `<PropertyName>` element refers to the value of a feature attribute. It contains a **string** or an **XPath expression** specifying the attribute name.

Literal

The `<Literal>` element specifies a constant value. It contains data of one of the following types:

Type	Description
Numeric	A string representing a numeric value (integer or decimal).
Boolean	A boolean value of `true` or `false`.
String	A string value. XML-incompatible text may be included by using **character entities** or `<![CDATA[ ]]>` delimiters.
Date	A string representing a date.
Geometry	An element specifying a geometry in GML3 format.

8.2.5 WFS 2.0 namespaces

WFS 2.0 does not depend on any one GML version and thus requires an explicit namespace and schemaLocation for GML. In a GET request, namespaces can be placed on a Filter element (that is, `filter=` the block below, URL-encoded):

```
<fes:Filter
        xmlns:fes="http://www.opengis.net/fes/2.0"
        xmlns:gml="http://www.opengis.net/gml/3.2">
    <fes:Not>
        <fes:Disjoint>
            <fes:ValueReference>sf:the_geom</fes:ValueReference>
            <gml:Polygon
                    gml:id="polygon.1"
                    srsName='http://www.opengis.net/def/crs/EPSG/0/26713'>
                <gml:exterior>
                    <gml:LinearRing>
                        <gml:posList>590431 4915204 590430
                            4915205 590429 4915204 590430
                            4915203 590431 4915204</gml:posList>
                    </gml:LinearRing>
                </gml:exterior>
            </gml:Polygon>
        </fes:Disjoint>
    </fes:Not>
</fes:Filter>
```

8.3 ECQL Reference

This section provides a reference for the syntax of the ECQL language. The full language grammar is documented in the the GeoTools ECQL BNF definition

8.3.1 Syntax Notes

The sections below describe the major language constructs. Each construct lists all syntax options for it. Each option is defined as a sequence of other constructs, or recursively in terms of itself.

- Symbols which are part of the ECQL language are shown in `code font`. All other symbols are part of the grammar description.
- ECQL keywords are not case-sensitive.
- A vertical bar symbol '|' indicates that a choice of keyword can be made.
- Brackets '[...]' delimit syntax that is optional.
- Braces '{ ... }' delimit syntax that may be present zero or more times.

8.3.2 Condition

A filter condition is a single predicate, or a logical combination of other conditions.

Syntax	Description
Predicate	Single predicate expression
Condition `AND` | `OR` *Condition*	Conjunction or disjunction of conditions
`NOT` *Condition*	Negation of a condition
`(` | `[` *Condition* `]` | `)`	Bracketing with `(` or `[` controls evaluation order

8.3.3 Predicate

Predicates are boolean-valued expressions which specify relationships between values.

Syntax	Description
Expression = | <> | < | <= | > | >= *Expression*	Comparison operations
Expression [`NOT`] `BETWEEN` *Expression* `AND` *Expression*	Tests whether a value lies in or outside a range (inclusive)
Expression [`NOT`] `LIKE` | `ILIKE` *like-pattern*	Simple pattern matching. *like-pattern* uses the `%` character as a wild-card for any number of characters. `ILIKE` does case-insensitive matching.
Attribute [`NOT`] `IN` `(` *Expression* { `,` *Expression* } `)`	Tests whether an expression value is (not) in a set of values
`IN` `(` *Literal* { `,` *Literal* } `)`	Tests whether a feature ID value is in a given set. ID values are integers or string literals
Expression `IS` [`NOT`] `NULL`	Tests whether a value is (non-)null
Attribute `EXISTS` | `DOES-NOT-EXIST`	Tests whether a featuretype does (not) have a given attribute
`INCLUDE` | `EXCLUDE`	Always include (exclude) features to which this filter is applied

Temporal Predicate

Temporal predicates specify the relationship of a time-valued expression to a time or time period.

Syntax	Description
Expression `BEFORE` *Time*	Tests whether a time value is before a point in time
Expression `BEFORE OR DURING` *Time Period*	Tests whether a time value is before or during a time period
Expression `DURING` *Time Period*	Tests whether a time value is during a time period
Expression `DURING OR AFTER` *Time Period*	Tests whether a time value is during or after a time period
Expression `AFTER` *Time*	Tests whether a time value is after a point in time

Spatial Predicate

Spatial predicates specify the relationship between geometric values. Topological spatial predicates (`INTERSECTS`, `DISJOINT`, `CONTAINS`, `WITHIN`, `TOUCHES` `CROSSES`, `OVERLAPS` and `RELATE`) are defined in terms of the DE-9IM model described in the OGC Simple Features for SQL specification.

Syntax	Description
`INTERSECTS(`*Expression* `,` *Expression* `)`	Tests whether two geometries intersect. The converse of `DISJOINT`
`DISJOINT(`*Expression* `,` *Expression* `)`	Tests whether two geometries are disjoint. The converse of `INTERSECTS`
`CONTAINS(`*Expression* `,` *Expression* `)`	Tests whether the first geometry topologically contains the second. The converse of `WITHIN`
`WITHIN(`*Expression* `,` *Expression* `)`	Tests whether the first geometry is topologically within the second. The converse of `CONTAINS`
`TOUCHES(`*Expression* `,` *Expression* `)`	Tests whether two geometries touch. Geometries touch if they have at least one point in common, but their interiors do not intersect.
`CROSSES(`*Expression* `,` *Expression* `)`	Tests whether two geometries cross. Geometries cross if they have some but not all interior points in common
`OVERLAPS(`*Expression* `,` *Expression* `)`	Tests whether two geometries overlap. Geometries overlap if they have the same dimension, have at least one point each not shared by the other, and the intersection of the interiors of the two geometries has the same dimension as the geometries themselves
`EQUALS(`*Expression* `,` *Expression* `)`	Tests whether two geometries are topologically equal
`RELATE(` *Expression* `,` *Expression* `,` *pattern* `)`	Tests whether geometries have the spatial relationship specified by a DE-9IM matrix *pattern*. A DE-9IM pattern is a string of length 9 specified using the characters `*TF012`. Example: `'1*T***T**'`
`DWITHIN(` *Expression* `,` *Expression* `,` *distance* `,` *units* `)`	Tests whether the distance between two geometries is no more than the specified distance. *distance* is an unsigned numeric value for the distance tolerance. *units* is one of `feet`, `meters`, `statute miles`, `nautical miles`, `kilometers`
`BEYOND(` *Expression* `,` *Expression* `,` *distance* `,` *units* `)`	Similar to `DWITHIN`, but tests whether the distance between two geometries is greater than the given distance.
`BBOX (` *Expression* `,` *Number* `,` *Number* `,` *Number* `,` *Number* [`,` *CRS*] `)`	Tests whether a geometry intersects a bounding box specified by its minimum and maximum X and Y values. The optional *CRS* is a string containing an SRS code (For example, `'EPSG:1234'`. The default is to use the CRS of the queried layer)

8.3.4 Expression

An expression specifies a attribute, literal, or computed value. The type of the value is determined by the nature of the expression. The standard PEMDAS order of evaluation is used.

Syntax	Description
Attribute	Name of a feature attribute
Literal	Literal value
Expression `+` | `-` | `*` | `/` *Expression*	Arithmetic operations
function `(` [*Expression* { `,` *Expression* }] `)`	Value computed by evaluation of a *filter function* with zero or more arguments.
`(` | `[` *Expression* `]` | `)`	Bracketing with `(` or `[` controls evaluation order

8.3.5 Attribute

An attribute name denotes the value of a feature attribute.

- Simple attribute names are sequences of letters and numbers,
- Attribute names quoted with double-quotes may be any sequence of characters.

8.3.6 Literal

Literals specify constant values of various types.

Type	Description
Number	Integer or floating-point number. Scientific notation is supported.
Boolean	`TRUE` or `FALSE`
String	String literal delimited by single quotes. To include a single quote in the string use two single-quotes: `''`
Geometry	Geometry in WKT format. WKT is defined in the OGC Simple Features for SQL specification. All standard geometry types are supported: `POINT`, `LINESTRING`, `POLYGON`, `MULTIPOINT`, `MULTILINESTRING`, `MULTIPOLYGON`, `GEOMETRYCOLLECTION`. A custom type of Envelope is also supported with syntax `ENVELOPE` `(` *x1 x2 y1 y2* `)`.
Time	A UTC date/time value in the format `yyyy-mm-hhThh:mm:ss`. The seconds value may have a decimal fraction. The time zone may be specified as `Z` or `+/-hh:mm`. Example: `2006-11-30T00:30:00Z`
Duration	A time duration specified as `P` [y `Y` m `M` d `D`] `T` [h `H` m `M` s `S`]. The duration can be specified to any desired precision by including only the required year, month, day, hour, minute and second components. Examples: `P1Y2M`, `P4Y2M20D`, `P4Y2M1DT20H3M36S`

Time Period

Specifies a period of time, in several different formats.

Syntax	Description
Time `/` *Time*	Period specified by a start and end time
Duration `/` *Time*	Period specified by a duration before a given time
Time `/` *Duration*	Period specified by a duration after a given time

8.4 Filter functions

The OGC Filter Encoding specification provides a generic concept of a *filter function*. A filter function is a named function with any number of arguments, which can be used in a filter expression to perform specific calculations. This provides much richer expressiveness for defining filters. Filter functions can be used in both the XML Filter Encoding language and the textual ECQL language, using the syntax appropriate to the language.

GeoServer provides many different kinds of filter functions, covering a wide range of functionality including mathematics, string formatting, and geometric operations. A complete list is provided in the *Filter Function Reference*.

Note: The Filter Encoding specification provides a standard syntax for filter functions, but does not mandate a specific set of functions. Servers are free to provide whatever functions they want, so some function expressions may work only in specific software.

8.4.1 Examples

The following examples show how filter functions are used. The first shows enhanced WFS filtering using the `geometryType` function. The second shows how to use functions in SLD to get improved label rendering.

WFS filtering

Let's assume we have a feature type whose geometry field, `geom`, can contain any kind of geometry. For a certain application we need to extract only the features whose geometry is a simple point or a multipoint. This can be done using a GeoServer-specific filter function named `geometryType`. Here is the WFS request including the filter function:

```
<wfs:GetFeature service="WFS" version="1.0.0"
  outputFormat="GML2"
  xmlns:wfs="http://www.opengis.net/wfs"
  xmlns:ogc="http://www.opengis.net/ogc"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.opengis.net/wfs
                      http://schemas.opengis.net/wfs/1.0.0/WFS-basic.xsd">
  <wfs:Query typeName="sf:archsites">
    <ogc:Filter>
       <ogc:PropertyIsEqualTo>
          <ogc:Function name="geometryType">
             <ogc:PropertyName>geom</ogc:PropertyName>
          </ogc:Function>
          <ogc:Literal>Point</ogc:Literal>
       </ogc:PropertyIsEqualTo>
    </ogc:Filter>
    </wfs:Query>
</wfs:GetFeature>
```

WFS 2.0 namespaces

WFS 2.0 does not depend on any one GML version and thus requires an explicit namespace and schemaLocation for GML. This POST example selects features using a spatial query. Note the complete declaration of namespace prefixes. In a GET request, namespaces can be placed on a Filter element.

```
<?xml version="1.0" encoding="UTF-8"?>
<wfs:GetFeature service="WFS" version="2.0.0"
    xmlns:wfs="http://www.opengis.net/wfs/2.0"
    xmlns:fes="http://www.opengis.net/fes/2.0"
    xmlns:gml="http://www.opengis.net/gml/3.2"
    xmlns:sf="http://www.openplans.org/spearfish"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.opengis.net/wfs/2.0
                        http://schemas.opengis.net/wfs/2.0/wfs.xsd
                        http://www.opengis.net/gml/3.2
                        http://schemas.opengis.net/gml/3.2.1/gml.xsd">
    <wfs:Query typeNames="sf:bugsites">
        <fes:Filter>
            <fes:Not>
                <fes:Disjoint>
                    <fes:ValueReference>sf:the_geom</fes:ValueReference>
                    <!-- gml:id is mandatory on GML 3.2 geometry elements -->
                    <gml:Polygon
                            gml:id="polygon.1"
                            srsName='http://www.opengis.net/def/crs/EPSG/0/26713'>
                        <gml:exterior>
                            <gml:LinearRing>
                                <!-- pairs must form a closed ring -->
                                <gml:posList>590431 4915204 590430
                                    4915205 590429 4915204 590430
                                    4915203 590431 4915204</gml:posList>
                            </gml:LinearRing>
                        </gml:exterior>
                    </gml:Polygon>
                </fes:Disjoint>
            </fes:Not>
        </fes:Filter>
    </wfs:Query>
</wfs:GetFeature>
```

SLD formatting

We want to display elevation labels in a contour map. The elevations are stored as floating point values, so the raw numeric values may display with unwanted decimal places (such as "150.0" or "149.999999"). We want to ensure the numbers are rounded appropriately (i.e. to display "150"). To achieve this the `numberFormat` filter function can be used in the SLD label content expression:

```
...
<TextSymbolizer>
  <Label>
    <ogc:Function name="numberFormat">
      <ogc:Literal>##</ogc:Literal>
      <ogc:PropertyName>ELEVATION</ogc:PropertyName>
    </ogc:Function>
  </Label>
```

```
  ...
</TextSymbolizer>
...
```

8.4.2 Performance implications

Using filter functions in SLD symbolizer expressions does not have significant overhead, unless the function is performing very heavy computation.

However, using functions in WFS filtering or SLD rule expressions may cause performance issues in certain cases. This is usually because specific filter functions are not recognized by a native data store filter encoder, and thus GeoServer must execute the functions in memory instead.

For example, given a filter like `BBOX(geom,-10,30,20,45) and geometryType(geom) = 'Point'` most data stores will split the filter into two separate parts. The bounding box filter will be encoded as a primary filter and executed in SQL, while the `geometryType` function will be executed in memory on the results coming from the primary filter.

8.5 Filter Function Reference

This reference describes all filter functions that can be used in WFS/WMS filtering or in SLD expressions.

The list of functions available on a GeoServer instance can be determined by browsing to http://localhost:8080/geoserver/wfs?request=GetCapabilities and searching for `ogc:FunctionNames` in the returned XML. If a function is described in the Capabilities document but is not in this reference, then it might mean that the function cannot be used for filtering, or that it is new and has not been documented. Ask for details on the user mailing list.

Unless otherwise specified, none of the filter functions in this reference are understood natively by the data stores, and thus expressions using them will be evaluated in-memory.

8.5.1 Function argument type reference

Type	Description
Double	Floating point number, 8 bytes, IEEE 754. Ranges from 4.94065645841246544e-324d to 1.79769313486231570e+308d
Float	Floating point number, 4 bytes, IEEE 754. Ranges from 1.40129846432481707e-45 to 3.40282346638528860e+38. Smaller range and less accurate than Double.
Integer	Integer number, ranging from -2,147,483,648 to 2,147,483,647
Long	Integer number, ranging from -9,223,372,036,854,775,808 to +9,223,372,036,854,775,807
Number	A numeric value of any type
Object	A value of any type
String	A sequence of characters
Timestamp	Date and time information

8.5.2 Comparison functions

Name	Arguments	Description
between	num:Number, low:Number, high:Number	returns true if low <= num <= high
equalTo	a:Object, b:Object	Can be used to compare for equality two numbers, two strings, two dates, and so on
greaterEqualThan	x:Object, y:Object	Returns true if x >= y. Parameters can be either numbers or strings (in the second case lexicographic ordering is used)
greaterThan	x:Object, y:Object	Returns true if x > y. Parameters can be either numbers or strings (in the second case lexicographic ordering is used)
in2, in3, in4, in5, in6, in7, in8, in9, in10	candidate:Object, v1:Object, ..., v9:Object	Returns true if candidate is equal to one of the v1, ..., v9 values. Use the function name matching the number of arguments specified.
in	candidate:Object, v1:Object, v2:Object, ...	Works exactly the same as the in2, ..., in10 functions described above, but takes any number of values as input.
isLike	string:String, pattern:String	Returns true if the string matches the specified pattern. For the full syntax of the pattern specification see the Java Pattern class javadocs
isNull	obj:Object	Returns true the passed parameter is null, false otherwise
lessThan	x:Object, y:Object	Returns true if x < y. Parameters can be either numbers or strings (in the second case lexicographic ordering is used
lessEqualThan	x:Object, y:Object	Returns true if x <= y. Parameters can be either numbers or strings (in the second case lexicographic ordering is used
not	bool:Boolean	Returns the negation of bool
notEqual	x:Object, y:Object	Returns true if x and y are equal, false otherwise

8.5.3 Control functions

Name	Arguments	Description
if_then_else	condition:Boolean, x:Object, y: Object	Returns x if the condition is true, y otherwise

8.5.4 Environment function

This function returns the value of environment variables defined in various contexts. Contexts which define environment variables include *SLD rendering* and the *WMS Animator*.

Name	Arguments	Description
env	variable:String	Returns the value of the environment variable variable.

8.5.5 Feature functions

Name	Arguments	Description
id	`feature`:Feature	returns the identifier of the feature
PropertyExists	`f`:Feature, `propertyName`:String	Returns `true` if `f` has a property named `propertyName`
property	`f`:Feature, `propertyName`:String	Returns the value of the property `propertyName`. Allows property names to be computed or specified by *Variable substitution in SLD*.

8.5.6 Spatial Relationship functions

For more information about the precise meaning of the spatial relationships consult the OGC Simple Feature Specification for SQL

Name	Arguments	Description
contains	`a`:Geometry, `b`:Geometry	Returns true if the geometry `a` contains `b`
crosses	`a`:Geometry, `b`:Geometry	Returns true if `a` crosses `b`
disjoint	`a`:Geometry, `b`:Geometry	Returns true if the two geometries are disjoint, false otherwise
equalsExact	`a`:Geometry, `b`:Geometry	Returns true if the two geometries are exactly equal, same coordinates in the same order
equalsExactTolerance	`a`:Geometry, `b`:Geometry, `tol`:Double	Returns true if the two geometries are exactly equal, same coordinates in the same order, allowing for a `tol` distance in the corresponding points
intersects	`a`:Geometry, `b`:Geometry	Returns true if `a` intersects `b`
isWithinDistance	`a`: Geometry, `b`:Geometry, `distance`: Double	Returns true if the distance between `a` and `b` is less than `distance` (measured as an euclidean distance)
overlaps	`a`: Geometry, `b`:Geometry	Returns true `a` overlaps with `b`
relate	`a`: Geometry, `b`:Geometry	Returns the DE-9IM intersection matrix for `a` and `b`
relatePattern	`a`: Geometry, `b`:Geometry, `pattern`:String	Returns true if the DE-9IM intersection matrix for `a` and `b` matches the specified pattern
touches	`a`: Geometry, `b`: Geometry	Returns true if `a` touches `b` according to the SQL simple feature specification rules
within	`a`: Geometry, `b`:Geometry	Returns true is fully contained inside `b`

8.5.7 Geometric functions

Name	Arguments	Description

Continued on next page

Table 8.1 – continued from previous page

area	`geometry`:Geometry	The area of the specified geometry. Works in a Cartesian plane, the result will be in the same unit of measure as the geometry coordinates (which also means the results won't make any sense for geographic data)
boundary	`geometry`:Geometry	Returns the boundary of a geometry
boundaryDimension	`geometry`:Geometry	Returns the number of dimensions of the geometry boundary
buffer	`geometry`:Geometry, `distance`:Double	Returns the buffered area around the geometry using the specified distance
bufferWithSegments	`geometry`:Geometry, `distance`:Double, `segments`:Integer	Returns the buffered area around the geometry using the specified distance and using the specified number of segments to represent a quadrant of a circle.
centroid	`geometry`:Geometry	Returns the centroid of the geometry. Can be often used as a label point for polygons, though there is no guarantee it will actually lie inside the geometry
convexHull	`geometry`:Geometry	Returns the convex hull of the specified geometry
difference	`a`:Geometry, `b`:Geometry	Returns all the points that sit in `a` but not in `b`
dimension	`a`:Geometry	Returns the dimension of the specified geometry
distance	`a`:Geometry, `b`:Geometry	Returns the euclidean distance between the two geometries
endAngle	`line`:LineString	Returns the angle of the end segment of the linestring
endPoint	`line`:LineString	Returns the end point of the linestring
envelope	`geometry`:geometry	Returns the polygon representing the envelope of the geometry, that is, the minimum rectangle with sides parallels to the axis containing it
exteriorRing	`poly`:Polygon	Returns the exterior ring of the specified polygon
geometryType	`geometry`:Geometry	Returns the type of the geometry as a string. May be `Point`, `MultiPoint`, `LineString`, `LinearRing`, `MultiLineString`, `Polygon`, `MultiPolygon`, `GeometryCollection`
geomFromWKT	`wkt`:String	Returns the `Geometry` represented in the Well Known Text format contained in the `wkt` parameter
geomLength	`geometry`:Geometry	Returns the length/perimeter of this geometry (computed in Cartesian space)
getGeometryN	`collection`:GeometryCollection, `n`:Integer	Returns the n-th geometry inside the collection
getX	`p`:Point	Returns the `x` ordinate of `p`
getY	`p`:Point	Returns the `y` ordinate of `p`
getZ	`p`:Point	Returns the `z` ordinate of `p`
interiorPoint	`geometry`:Geometry	Returns a point that is either interior to the geometry, when possible, or sitting on its boundary, otherwise
interiorRingN	`polyg`:Polygon, `n`:Integer	Returns the n-th interior ring of the polygon
intersection	`a`:Geometry, `b`:Geometry	Returns the intersection between `a` and `b`. The intersection result can be anything including a geometry collection of heterogeneous, if the result is empty, it will be represented by an empty collection.
isClosed	`line`: LineString	Returns true if `line` forms a closed ring, that is, if the first and last coordinates are equal

Continued on next page

Table 8.1 – continued from previous page

isEmpty	`geometry`:Geometry	Returns true if the geometry does not contain any point (typical case, an empty geometry collection)
isometric	`geometry`:Geometry, `extrusion`:Double	Returns a MultiPolygon containing the isometric extrusions of all components of the input geometry. The extrusion distance is `extrusion`, expressed in the same unit as the geometry coordinates. Can be used to get a pseudo-3d effect in a map
isRing	`line`:LineString	Returns true if the `line` is actually a closed ring (equivalent to `isRing(line) and isSimple(line)`)
isSimple	`line`:LineString	Returns true if the geometry self intersects only at boundary points
isValid	`geometry`: Geometry	Returns true if the geometry is topologically valid (rings are closed, holes are inside the hull, and so on)
numGeometries	`collection`: GeometryCollection	Returns the number of geometries contained in the geometry collection
numInteriorRing	`poly`: Polygon	Returns the number of interior rings (holes) inside the specified polygon
numPoint	`geometry`: Geometry	Returns the number of points (vertexes) contained in `geometry`
offset	`geometry`: Geometry, `offsetX`:Double, `offsetY`:Double	Offsets all points in a geometry by the specified X and Y offsets. Offsets are working in the same coordinate system as the geometry own coordinates.
pointN	`geometry`: Geometry, `n`:Integer	Returns the n-th point inside the specified geometry
startAngle	`line`: LineString	Returns the angle of the starting segment of the input linestring
startPoint	`line`: LineString	Returns the starting point of the input linestring
symDifference	`a`: Geometry, `b`:Geometry	Returns the symmetrical difference between `a` and `b` (all points that are inside `a` or `b`, but not both)
toWKT	`geometry`: Geometry	Returns the WKT representation of `geometry`
union	`a`: Geometry, `b`:Geometry	Returns the union of `a` and `b` (the result may be a geometry collection)
vertices	`geom`: Geometry	Returns a multi-point made with all the vertices of `geom`

8.5.8 Math functions

Name	Arguments	Description
abs	`value`:Integer	The absolute value of the specified Integer `value`
abs_2	`value`:Long	The absolute value of the specified Long `value`
abs_3	`value`:Float	The absolute value of the specified Float `value`
abs_4	`value`:Double	The absolute value of the specified Double `value`
acos	`angle`:Double	Returns the arc cosine of an `angle` in radians, in the range of 0.0 through `PI`
asin	`angle`:Double	Returns the arc sine of an `angle` in radians, in the range of `-PI / 2` through `PI / 2`
atan	`angle`:Double	Returns the arc tangent of an angle in radians, in the range of `-PI/2` through `PI/2`
atan2	`x`:Double, `y`:Double	Converts a rectangular coordinate `(x, y)` to polar **(r, theta)** and returns **theta**.

Continued on next page

Table 8.2 – continued from previous page

ceil	x: Double	Returns the smallest (closest to negative infinity) double value that is greater than or equal to x and is equal to a mathematical integer.
cos	angle: Double	Returns the cosine of an angle expressed in radians
double2bool	x: Double	Returns true if x is zero, false otherwise
exp	x: Double	Returns Euler's number **e** raised to the power of x
floor	x: Double	Returns the largest (closest to positive infinity) value that is less than or equal to x and is equal to a mathematical integer
IEEERemainder	x: Double, y:Double	Computes the remainder of x divided by y as prescribed by the IEEE 754 standard
int2bbool	x: Integer	Returns true if x is zero, false otherwise
int2ddouble	x: Integer	Converts x to a Double
log	x: Integer	Returns the natural logarithm (base e) of x
max, max_3, max_4	x1: Double, x2:Double, x3:Double, x4:Double	Returns the maximum between x1, ..., x4
min, min_3, min_4	x1: Double, x2:Double, x3:Double, x4:Double	Returns the minimum between x1, ..., x4
pi	None	Returns an approximation of pi, the ratio of the circumference of a circle to its diameter
pow	base:Double, exponent:Double	Returns the value of base raised to the power of exponent
random	None	Returns a Double value with a positive sign, greater than or equal to 0.0 and less than 1.0. Returned values are chosen pseudo-randomly with (approximately) uniform distribution from that range.
rint	x:Double	Returns the Double value that is closest in value to the argument and is equal to a mathematical integer. If two double values that are mathematical integers are equally close, the result is the integer value that is even.
round_2	x:Double	Same as round, but returns a Long
round	x:Double	Returns the closest Integer to x. The result is rounded to an integer by adding 1/2, taking the floor of the result, and casting the result to type Integer. In other words, the result is equal to the value of the expression (int)floor(a + 0.5)
roundDouble	x:Double	Returns the closest Long to x
sin	angle: Double	Returns the sine of an angle expressed in radians
tan	angle:Double	Returns the trigonometric tangent of angle expressed in radians
toDegrees	angle:Double	Converts an angle expressed in radians into degrees
toRadians	angle:Double	Converts an angle expressed in radians into degrees

8.5.9 String functions

String functions generally will accept any type of value for String arguments. Non-string values will be converted into a string representation automatically.

Name	Arguments	Description
Concatenate	`s1`:String, `s2`:String, ...	Concatenates any number of strings. Non-string arguments are allowed.
strAbbreviate	`sentence`:String, `lower`:Integer, `upper`:Integer, `append`:String	Abbreviates the sentence at first space beyond `lower` (or at `upper` if no space). Appends `append` if string is abbreviated.
strCapitalize	`sentence`:String	Fully capitalizes the sentence. For example, "HoW aRe YOU?" will be turned into "How Are You?"
strConcat	`a`:String, `b`:String	Concatenates the two strings into one
strDefaultIfBlank	`str`:String, `default`:String	returns `default` if `str` is empty, blank or null
strEndsWith	`string`:String, `suffix`:String	Returns true if `string` ends with `suffix`
strEqualsIgnoreCase	`a`:String, `b`:String	Returns true if the two strings are equal ignoring case considerations
strIndexOf	`string`:String, `substring`:String	Returns the index within this string of the first occurrence of the specified substring, or `-1` if not found
strLastIndexOf	`string`:String, `substring`:String	Returns the index within this string of the last occurrence of the specified substring, or `-1` if not found
strLength	`string`:String	Returns the string length
strMatches	`string`:String, `pattern`:String	Returns true if the string matches the specified regular expression. For the full syntax of the pattern specification see the Java Pattern class javadocs
strReplace	`string`:String, `pattern`:String, `replacement`:String, `global`: boolean	Returns the string with the pattern replaced with the given replacement text. If the `global` argument is `true` then all occurrences of the pattern will be replaced, otherwise only the first. For the full syntax of the pattern specification see the Java Pattern class javadocs
strStartsWith	`string`:String, `prefix`:String	Returns true if `string` starts with `prefix`
strStripAccents	`string`:String	Removes diacritics (~= accents) from a string. The case will not be altered.
strSubstring	`string`:String, `begin`:Integer, `end`:Integer	Returns a new string that is a substring of this string. The substring begins at the specified `begin` and extends to the character at index `endIndex - 1` (indexes are zero-based).
strSubstringStart	`string`:String, `begin`:Integer	Returns a new string that is a substring of this string. The substring begins at the specified `begin` and extends to the last character of the string
strToLowerCase	`string`:String	Returns the lower case version of the string
strToUpperCase	`string`:String	Returns the upper case version of the string
strTrim	`string`:String	Returns a copy of the string, with leading and trailing white space omitted

8.5.10 Parsing and formatting functions

Name	Arguments	Description
dateFormat	`format`:String, `date`:Timestamp	Formats the specified date according to the provided format. The format syntax can be found in the Java SimpleDateFormat javadocs
dateParse	`format`:String, `dateString`:String	Parses a date from a `dateString` formatted according to the `format` specification. The format syntax can be found in the Java SimpleDateFormat javadocs
numberFormat	`format`:String, `number`:Double	Formats the number according to the specified `format`. The format syntax can be found in the Java DecimalFormat javadocs
parseBoolean	`boolean`:String	Parses a string into a boolean. The empty string, `f`, `0.0` and `0` are considered false, everything else is considered true.
parseDouble	`number`:String	Parses a string into a double. The number can be expressed in normal or scientific form.
parseInt	`number`:String	Parses a string into an integer.
parseLong	`number`:String	Parses a string into a long integer

8.5.11 Transformation functions

Transformation functions transform values from one data space into another. These functions provide a concise way to compute styling parameters from feature attribute values. See also *Styling using Transformation Functions*.

Name	Arguments	Description
Recode	`lookupValue`:Object, `data`:Object, `value`:Object, ...	Transforms a `lookupValue` from a set of discrete data values into another set of values. Any number of `data`/`value` pairs may be specified.
Categorize	`lookupValue`:Object, `value`:Object, `threshold`:Object, ... `value`:Object, `belongsTo` : String	Transforms a continuous-valued attribute value into a set of discrete values. `lookupValue` and `value` must be an orderable type (typically numeric). The initial `value` is required. Any number of additional `threshold`/`value` pairs may be specified. `belongsTo` is optional, with the value `succeeding` or `preceding`. It defines which interval to use when the lookup value equals a threshold value.
Interpolate	`lookupValue`:Numeric, `data`:Numeric, `value`:Numeric *or* #RRGGBB, ... `mode`:String, `method`:String	Transforms a continuous-valued attribute value into another continuous range of values. Any number of `data`/`value` pairs may be specified. `mode` is optional, with the value `linear`, `cosine` or `cubic`. It defines the interpolation algorithm to use. `method` is optional, with the value `numeric` or `color`. It defines whether the target values are numeric or RGB color specifications.

CHAPTER 9

Server configuration

This page will detail how to set various configuration options in GeoServer as well as test out the services in GeoServer.

9.1 Status

The Server Status page has two tabs to summarize the current status of GeoServer. The Status tab provides a summary of server configuration parameters and run-time status. The modules tab provides the status of the various modules installed on the server. This page provides a useful diagnostic tool in a testing environment.

9.1.1 Server Status

Status Field Descriptions

The following table describes the current status indicators.

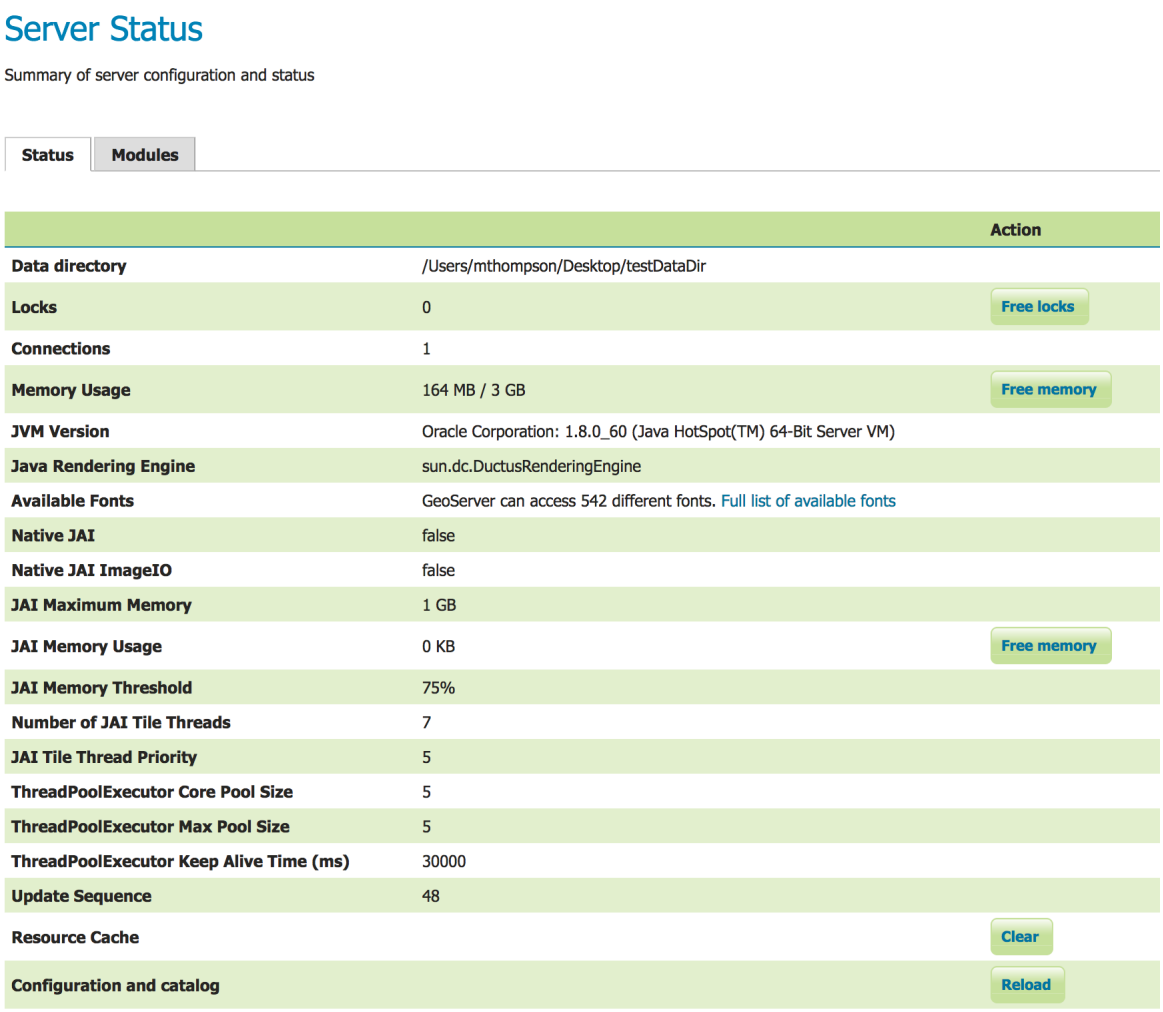


Fig. 9.1: Status Page (default tab)

Option	Description
Data directory	Shows the path to the GeoServer data directory (GEOSERVER_DATA_DIR property).
Locks	A WFS has the ability to lock features to prevent more than one person from updating the feature at one time. If data is locked, edits can be performed by a single WFS editor. When the edits are posted, the locks are released and features can be edited by other WFS editors. A zero in the locks field means all locks are released. If locks is non-zero, then pressing "free locks," releases all feature locks currently help by the server, and updates the field value to zero.
Connections	Refers to the numbers of vector stores, in the above case 4, that were able to connect.
Memory Usage	The amount of memory current used by GeoServer. In the above example, 118 MB of memory out of a total of 910 MB is being used. Clicking on the "Free Memory" button, cleans up memory marked for deletion by running the garbage collector.
JVM Version	Denotes which version of the JVM (Java Virtual Machine) is been used to power the server. Here the JVM is Oracle Corporation.: 1.8.0_60.
Java Rendering Engine	Shows the rendering engine used for vector operations.
Available Fonts	Shows the number of fonts available. Selecting the link will show the full list.
Native JAI	GeoServer uses Java Advanced Imaging (JAI) framework for image rendering and coverage manipulation. When properly installed (true), JAI makes WCS and WMS performance faster and more efficient.
Native JAI ImageIO	GeoServer uses JAI Image IO (JAI) framework for raster data loading and image encoding. When properly installed (true), JAI Image I/O makes WCS and WMS performance faster and more efficient.
JAI Maximum Memory	Expresses in bytes the amount of memory available for tile cache, in this case 455 Mbytes.
JAI Memory Usage	Run-time amount of memory is used for the tile cache. Clicking on the "Free Memory" button, clears available JAI memory by running the tile cache flushing.
JAI Memory Threshold	Refers to the percentage, e.g. 75, of cache memory to retain during tile removal.
Number of JAI Tile Threads	The number of parallel threads used by the scheduler to handle tiles.
JAI Tile Thread Priority	Schedules the global tile scheduler priority. The priority value defaults to 5, and must fall between 1 and 10.
ThreadPoolExecutor Core Pool Size	Number of threads that the ThreadPoolExecutor will create. This is underlying Java runtime functionality - see the Java documentation for ThreadPoolExecutor for more information.
ThreadPoolExecutor Max Pool Size	Maximum number of threads that the ThreadPoolExecutor will create. This is underlying Java runtime functionality - see the Java documentation for ThreadPoolExecutor for more information.
ThreadPoolExecutor Keep Alive Time (ms)	Timeout for threads to be terminated if they are idle and more than the core pool number exist. This is underlying Java runtime functionality - see the Java documentation for ThreadPoolExecutor for more information.
Update Sequence	Refers to the number of times (426) the server configuration has been modified.
Resource cache	GeoServer does not cache data, but it does cache connection to stores, feature type definitions, external graphics, font definitions and CRS definitions as well. The "Clear" button forces those caches empty and makes GeoServer reopen the stores and re-read image and font information, as well as the custom CRS definitions stored in *${GEOSERVER_DATA_DIR}/user_projections/epsg.properties*.
Configuration and catalog	GeoServer keeps in memory all of its configuration data. If for any reason that configuration information has become stale (e.g., an external utility has modified the configuration on disk) the "Reload" button will force GeoServer to reload all of its configuration from disk.

9.1.2 Module Status

The modules tab provides a summary of the status of all installed modules in the running server.

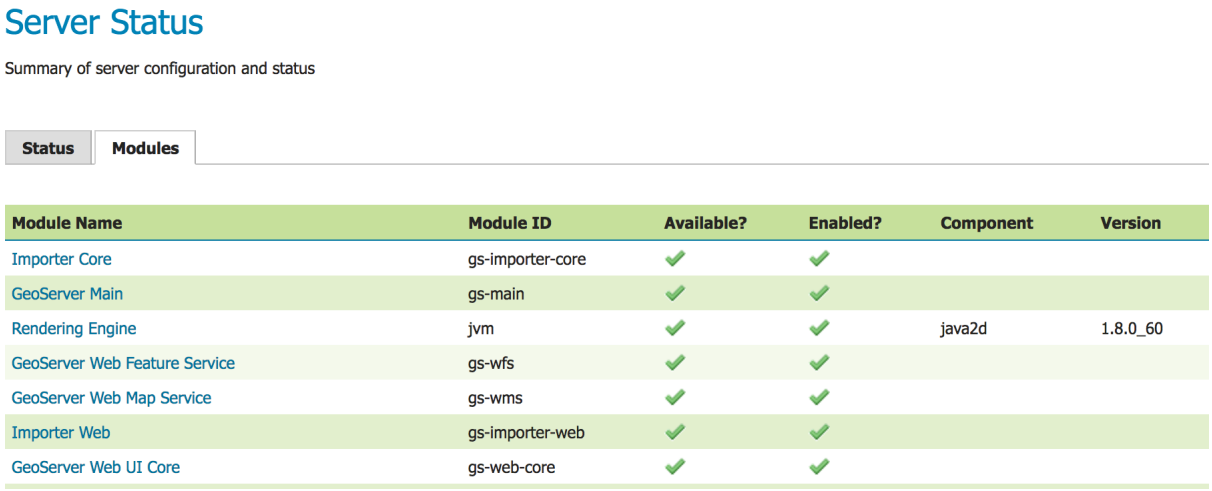


Fig. 9.2: Module Status

Module Status Field Descriptions

Module Name	The human readable name of the module, this links to a popup containing the full details and messages of the module
Module ID	The internal package name of the module
Available?	Whether the module is available to GeoServer
Enabled?	Whether the module is enabled in the current GeoServer configuration
Component	(Optional) Optional component identifier within the module
Version	(Optional) The version of the installed module
Message (popup)	(Optional) status message such as what Java rendering engine is in use, or the library path if the module/driver is unavailable

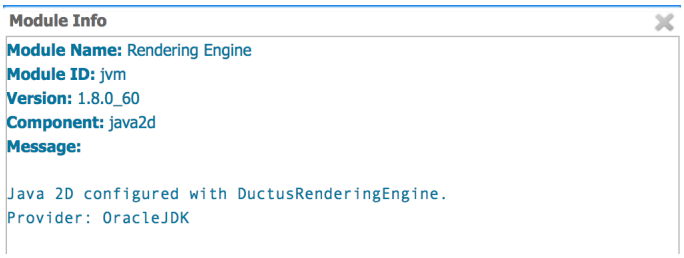


Fig. 9.3: Module Status popup

9.2 Contact Information

The Contact Information is used in the Capabilities document of the WMS server, and is publicly accessible. Please complete this form with the relevant information.

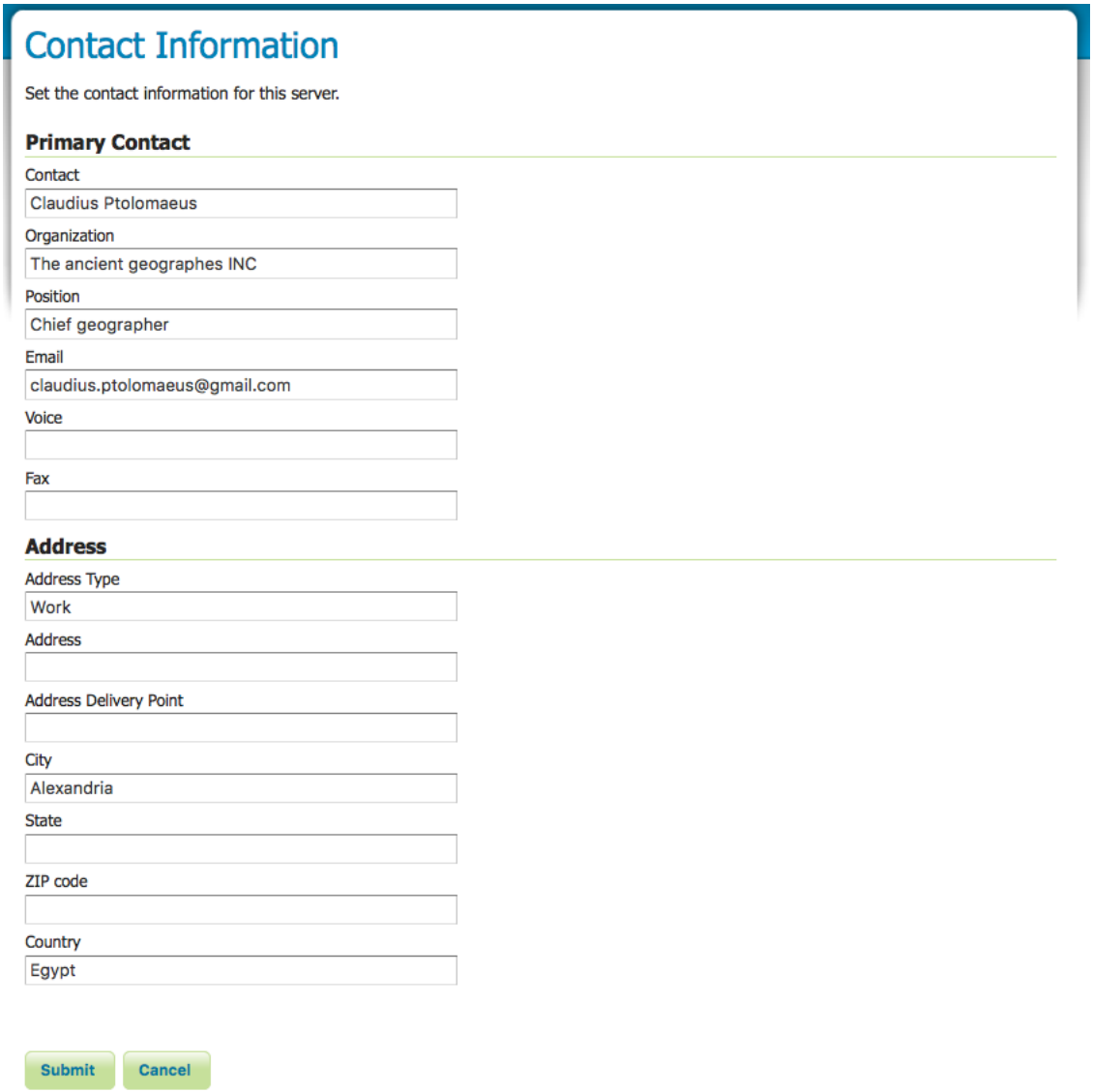


Fig. 9.4: Contact Page

9.2.1 Contact Information Fields

Field	Description
Contact	Contact information for webmaster
Organization	Name of the organization with which the contact is affiliated
Position	Position of the contact within their organization
Email	Contact email address
Telephone	Contact phone number
Fax	Contact Fax number
Address Type	Type of address specified, such as postal
Address	Actual street address
City	City of the address
State	State or province of the address
Zip code	Postal code for the address
Country	Country of the address

9.3 Global Settings

The Global Setting page configures messaging, logging, character, and proxy settings for the entire server. Global Settings are used to configure how OGC Web Services function.

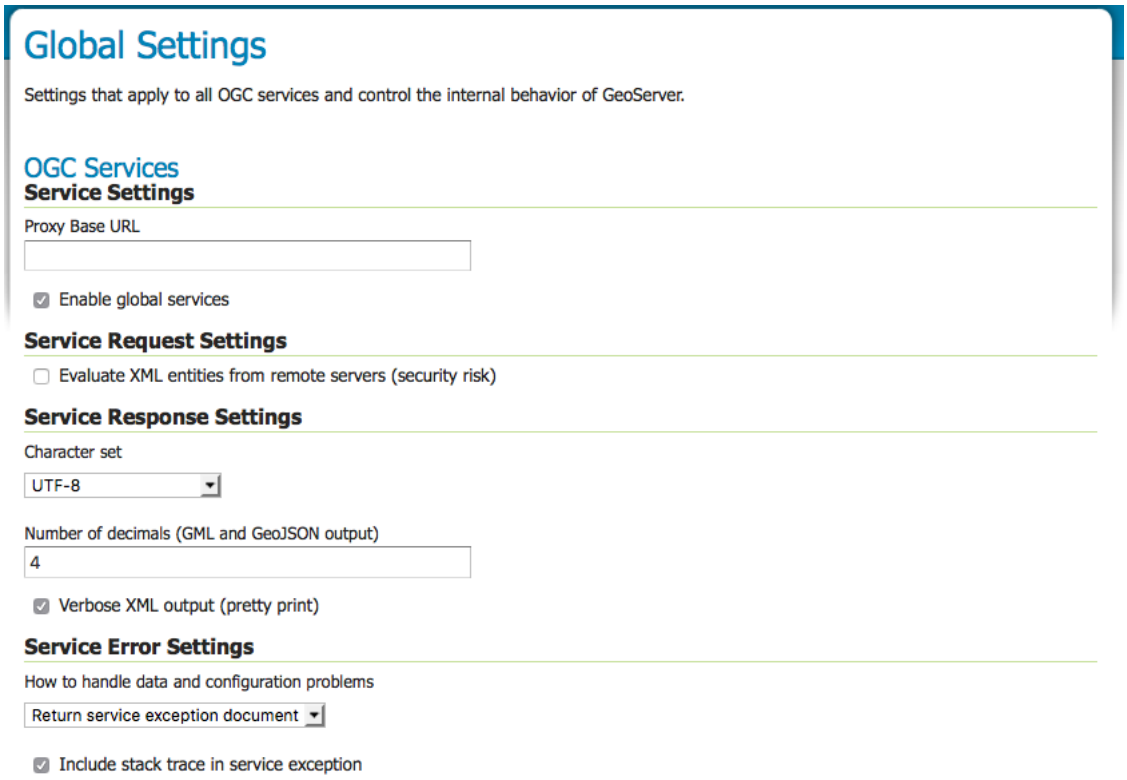


Fig. 9.5: Global Settings Service Configuration

Global Settings are also used to control the GeoServer application as a whole.

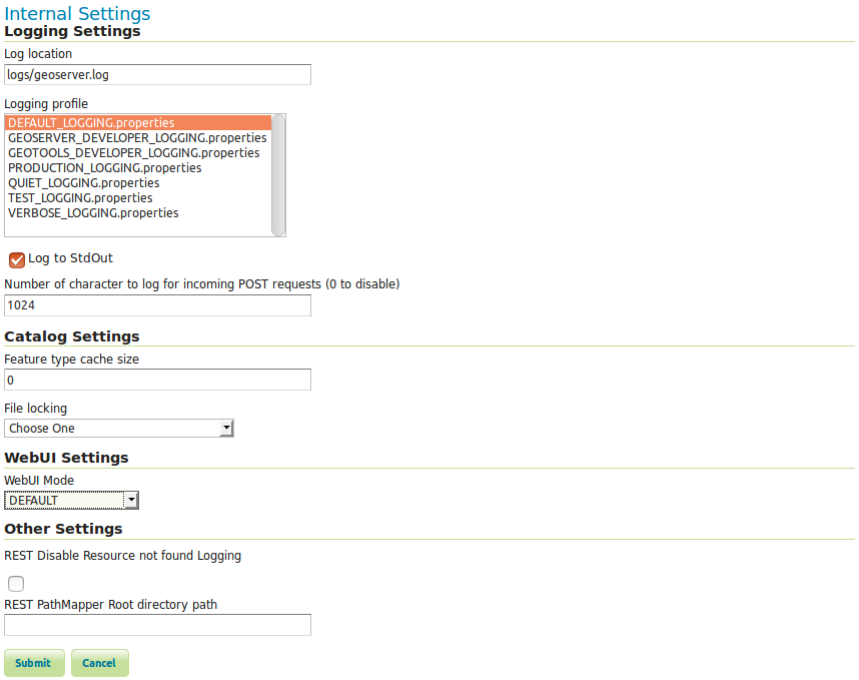


Fig. 9.6: Global Settings Internal Configuration

9.3.1 Verbose Messages

Verbose Messages, when enabled, will cause GeoServer to return XML with newlines and indents. Because such XML responses contain a larger amount of data, and in turn requires a larger amount of bandwidth, it is recommended to use this option only for testing purposes.

9.3.2 Verbose Exception Reporting

Verbose Exception Reporting returns service exceptions with full Java stack traces. It writes to the GeoServer log file and offers one of the most useful configuration options for debugging. When disabled, GeoServer returns single-line error messages.

9.3.3 Enable Global Services

When enabled, allows access to both global services and *virtual services*. When disabled, clients will only be able to access virtual services. Disabling is useful if GeoServer is hosting a large amount of layers and you want to ensure that client always request limited layer lists. Disabling is also useful for security reasons.

9.3.4 Handle data and configuration problems

This setting determines how GeoServer will respond when a layer becomes inaccessible for some reason. By default, when a layer has an error (for example, when the default style for the layer is deleted), a service exception is printed as part of the capabilities document, making the document invalid. For clients that rely on a valid capabilities document, this can effectively make a GeoServer appear to be "offline".

An administrator may prefer to configure GeoServer to simply omit the problem layer from the capabilities document, thus retaining the document integrity and allowing clients to connect to other published layers.

There are two options:

OGC_EXCEPTION_REPORT: This is the default behavior. Any layer errors will show up as Service Exceptions in the capabilities document, making it invalid.

SKIP_MISCONFIGURED_LAYERS: With this setting, GeoServer will elect simply to not describe the problem layer at
This is the default setting starting with GeoServer 2.11 and allows for faster startups, as the stores connectivity does not need to be checked in advance.

9.3.5 Number of Decimals

Refers to the number of decimal places returned in a GetFeature response. Also useful in optimizing bandwidth. Default is **8**.

9.3.6 Character Set

Specifies the global character encoding that will be used in XML responses. Default is **UTF-8**, which is recommended for most users. A full list of supported character sets is available on the IANA Charset Registry.

9.3.7 Proxy Base URL

GeoServer can have the capabilities documents report a proxy properly. "The Proxy Base URL" field is the base URL seen beyond a reverse proxy.

9.3.8 Use headers for Proxy URL

Checking this box allows a by-request modification of the proxy URL using templates (templates based on HTTP proxy headers).

The supported proxy headers are:

1. **X-Forwarded-Proto** The protocol used by the request
2. **X-Forwarded-Host** The hostname and port of the proxy URL
3. **X-Forwarded-For** The client IP address
4. **X-Forwarded-Path** The path of the proxy URL (this is not an official HTTP header, although it is supported by some web-servers)
5. **Forwarded** Header that supersedes the "X-Forwarded-*" headers above. It has these components: "by", "for", "host", "proto", "path" (this component is not official, but added for consistency with `X-Forwarded-Path`)
6. **Host** Same as `X-Forwarded`

For instance, to allow different protocols (`http` and `https`) and different hostnames, the proxy base URL field may be changed to: `${X-Forwarded-Proto}://${X-Forwarded-Host}/geoserver` The use of the `Forwarded` header is a tad more complex, as its components have to be referenced in templates with the dot-notation, as in: `{Forwarded.proto}://${Forwarded.host}/geoserver`.

Multiple templates can be put into the "Proxy Base URL". These templates provide fall-backs, since only the first one that is fully matched is used. For instance, a Proxy Base URL

of `http://${X-Forwarded-Host}/geoserver http://www.foo.org/geoserver` (Templates are space-separated.) can result in either: `http://www.example.com/geoserver` (if `X-Forwarded-Host` is set to `www.example.com`.) or `http://www.foo.org/geoserver` (if `X-Forwarded-Host` is not set.)

Header names in templates are case-insensitive.

9.3.9 Logging Profile

Logging Profile corresponds to a log4j configuration file in the GeoServer data directory. (Apache log4j is a Java-based logging utility.) By default, there are five logging profiles in GeoServer; additional customized profiles can be added by editing the log4j file.

There are six logging levels used in the log itself. They range from the least serious TRACE, through DEBUG, INFO, WARN, ERROR and finally the most serious, FATAL. The GeoServer logging profiles combine logging levels with specific server operations. The five pre-built logging profiles available on the global settings page are:

1. **Default Logging** (`DEFAULT_LOGGING`)—Provides a good mix of detail without being VERBOSE. Default logging enables INFO on all GeoTools and GeoServer levels, except certain (chatty) GeoTools packages which require WARN.
2. **GeoServer Developer Logging** (`GEOSERVER_DEVELOPER_LOGGING`)-A verbose logging profile that includes DEBUG information on GeoServer and VFNY. This developer profile is recommended for active debugging of GeoServer.
3. **GeoTools Developer Logging** (`GEOTOOLS_DEVELOPER_LOGGING`)—A verbose logging profile that includes DEBUG information only on GeoTools. This developer profile is recommended for active debugging of GeoTools.
4. **Production Logging** (`PRODUCTION_LOGGING`) is the most minimal logging profile, with only WARN enabled on all GeoTools and GeoServer levels. With such production level logging, only problems are written to the log files.
5. **Verbose Logging** (`VERBOSE_LOGGING`)—Provides more detail by enabling DEBUG level logging on GeoTools, GeoServer, and VFNY.

9.3.10 Log to StdOut

Standard output (StdOut) determines where a program writes its output data. In GeoServer, the Log to StdOut setting enables logging to the text terminal that initiated the program. If you are running GeoServer in a large J2EE container, you might not want your container-wide logs filled with GeoServer information. Clearing this option will suppress most GeoServer logging, with only FATAL exceptions still output to the console log.

9.3.11 Log Location

Sets the written output location for the logs. A log location may be a directory or a file, and can be specified as an absolute path (e.g., `C:\GeoServer\GeoServer.log`) or a relative one (for example, `GeoServer.log`). Relative paths are relative to the GeoServer data directory. Default is `logs/geoserver.log`.

9.3.12 XML POST request log buffer

In more verbose logging levels, GeoServer will log the body of XML (and other format) POST requests. It will only log the initial part of the request though, since it has to store (buffer) everything that gets logged

for use in the parts of GeoServer that use it normally. This setting sets the size of this buffer, in characters. A setting of **0** will disable the log buffer.

9.3.13 XML Entities

XML Requests sent to GeoServer can include references to other XML documents. Since these files are processed by GeoServer the facility could be used to access files on the server.

This option is only useful with the application schema extensions.

9.3.14 Feature type cache size

GeoServer can cache datastore connections and schemas in memory for performance reasons. The cache size should generally be greater than the number of distinct featuretypes that are expected to be accessed simultaneously. If possible, make this value larger than the total number of featuretypes on the server, but a setting too high may produce out-of-memory errors. On the other hand, a value lower than the total number of your registered featuretypes may clear and reload the resource-cache more often, which can be expensive and e.g. delay WFS-Requests in the meantime. The default value for the Feature type cache size is 100.

9.3.15 File Locking

This configuration settings allows control of they type of file locking used when accessing the GeoServer Data Directory. This setting is used to protected the GeoServer configuration from being corrupted by multiple parties editing simultaneously. File locking should be employed when using the REST API to configure GeoServer, and can protected GeoServer when more than one administrator is making changes concurrently.

There are three options:

NIO File locking: Uses Java New IO File Locks suitable for use in a clustered environment (with multiple GeoServers sharing the same data directory).

In-process locking: Used to ensure individual configuration files cannot be modified by two web administration or REST sessions at the same time.

Disable Locking: No file locking is used.

9.3.16 Web/UI Mode

This configuration setting allows control over WebUI redirecting behaviour. By default, when the user loads a page that contains input, a HTTP 302 Redirect response is returned that causes a reload of that same with a generated session ID in the request parameter. This session ID allows the state of the page to be remembered after a refresh and prevents any occurence of the 'double submit problem'. However, this behaviour is incompatible with clustering of multiple geoserver instances.

There are three options:

DEFAULT: Use redirecting unless a clustering module has been loaded.

REDIRECT: Always use redirecting (incompatible with clustering).

DO_NOT_REDIRECT: Never use redirecting (does not remember state when reloading a page and may cause double submit).

Note that a restart of GeoServer is necessary for a change in the setting to have effect.

9.3.17 REST Disable Resource not found Logging

This parameter can be used to mute exception logging when doing REST operations and the requested Resource is not present. This default setting can be overridden by adding to a REST call the following parameter: **quietOnNotFound=true/false**.

9.3.18 REST PathMapper Root directory path

This parameter is used by the RESTful API as the *Root Directory* for the newly uploaded files, following the structure:

```
${rootDirectory}/workspace/store[/<file>]
```

9.4 Image Processing

Java Advanced Imaging (JAI) is an image processing library built by Sun Microsystems. JAI Image I/O Tools provides reader, writer, and stream plug-ins for the standard Java Image I/O Framework.

Several JAI parameters, used by both WMS and WCS operations, can be configured in the Image Processing page.

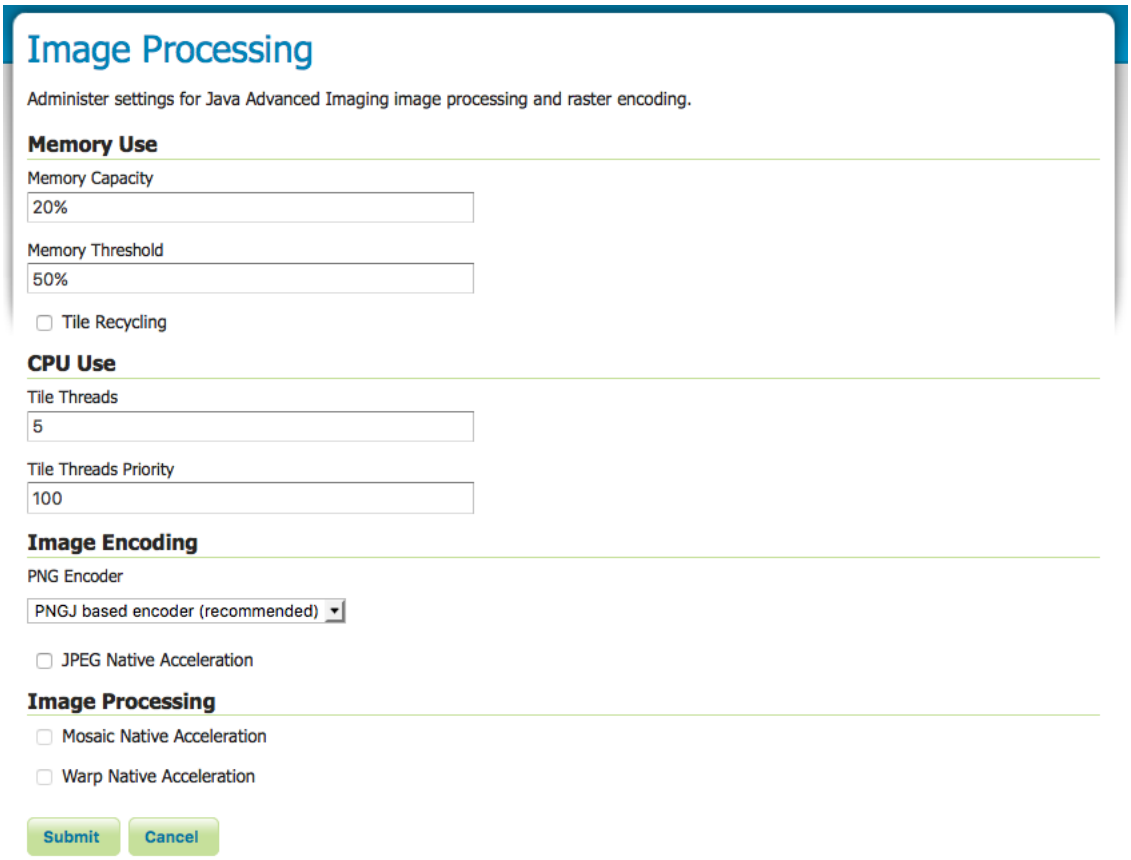


Fig. 9.7: Image Processing

9.4.1 Memory & Tiling

When supporting large images it is efficient to work on image subsets without loading everything to memory. A widely used approach is tiling which basically builds a tessellation of the original image so that image data can be read in parts rather than whole. Since very often processing one tile involves surrounding tiles, tiling needs to be accompanied by a tile-caching mechanism. The following JAI parameters allow you to manage the JAI cache mechanism for optimized performance.

Memory Capacity—For memory allocation for tiles, JAI provides an interface called TileCache. Memory Capacity sets the global JAI TileCache as a percentage of the available heap. A number between 0 and 1 exclusive. If the Memory Capacity is smaller than the current capacity, the tiles in the cache are flushed to achieve the desired settings. If you set a large amount of memory for the tile cache, interactive operations are faster but the tile cache fills up very quickly. If you set a low amount of memory for the tile cache, the performance degrades.

Memory Threshold—Sets the global JAI TileCache Memory threshold. Refers to the fractional amount of cache memory to retain during tile removal. JAI Memory Threshold value must be between 0.0 and 1.0. The Memory Threshold visible on the *Status* page.

Tile Threads—JAI utilizes a TileScheduler for tile calculation. Tile computation may make use of multithreading for improved performance. The Tile Threads parameter sets the TileScheduler, indicating the number of threads to be used when loading tiles.

Tile Threads Priority—Sets the global JAI Tile Scheduler thread priorities. Values range from 1 (Min) to 10 (Max), with default priority set to 5 (Normal).

Tile Recycling—Enable/Disable JAI Cache Tile Recycling. If selected, Tile Recycling allows JAI to re-use already loaded tiles, which can provide significant performance improvement.

Native Acceleration—To improve the computation speed of image processing applications, the JAI comes with both Java Code and native code for many platform. If the Java Virtual Machine (JVM) finds the native code, then that will be used. If the native code is not available, the Java code will be used. As such, the JAI package is able to provide optimized implementations for different platforms that can take advantage of each platform's capabilities.

JPEG Native Acceleration—Enables/disable JAI JPEG Native Acceleration. When selected, enables JPEG native code, which may speed performance, but compromise security and crash protection.

PNG Encoder Type—Provides a selection of the PNG encoder between the Java own encoder, the JAI ImageIO native one, and a PNGJ based one:

- The Java standard encoder is always set to maximum compression. It provides the smallest output images, balanced by a high performance cost (up to six times slower than the other two alternatives).
- The ImageIO native encoder, available only when the ImageIO native extensions are installed, provided higher performance, but also generated significantly larger PNG images
- The PNGJ based encoder provides the best performance and generated PNG images that are just slightly larger than the Java standard encoder. It is the recommended choice, but it's also newer than the other two, so in case of misbehavior the other two encoders are left as an option for the administrator.

Mosaic Native Acceleration—To reduce the overhead of handling them, large data sets are often split into smaller chunks and then combined to create an image mosaic. An example of this is aerial imagery which usually comprises thousands of small images at very high resolution. Both native and JAI implementations of mosaic are provided. When selected, Mosaic Native Acceleration use the native implementation for creating mosaics.

Warp Native Acceleration—Also for the Warp operation are provided both native and JAI implementations. If the checkbox is enabled, then the native operation is used for the warp operation.

It is quite important to remember that faster encoders are not necessarily going to visibly improve performance, if data loading and processing/rendering are dominating the response time, choosing a better encoder will likely not provide the expected benefits.

9.4.2 JAI-EXT

The JAI-EXT library is open-source project which aims to replace closed source JAI project provided by Sun. The main difference between *JAI* and *JAI-EXT* operations is the support for external **Region of Interests** (ROI) and image **NoData** in *JAI-EXT*.

The following panel is be available at the bottom of the JAI Settings page

Fig. 9.8: JAI/JAIEXT Setup panel

This panel can be used to selectively enable/disable JAI-EXT operations in favor of JAI ones.

By default, **JAI-EXT** operations are enabled. In case of misbehavior add the following java option to GeoServer startup script and restart GeoServer to disable them.

```
-Dorg.geotools.coverage.jaiext.enabled=false
```

Warning: Users should take care that *JAI* native libraries are not supported by *JAI-EXT*, since *JAI-EXT* is a pure Java API.

9.5 Raster Access

The Coverage Access Settings page in the Server menu in the *Web administration interface* provides configuration options to customize thread pool executors and ImageIO caching memory.

9.5.1 Memory Use

WMS requests usually produce relatively small images whilst WCS requests may frequently deal with bigger datasets. Caching the image in memory before encoding it may be helpful when the size of the image isn't too big. For a huge image (as one produced by a big WCS request) it would be better instead caching through a temporary file with respect to caching in memory. This section allows to specify a threshold image size to let GeoServer decide whether to use a MemoryCacheImageOutputStream or FileCacheImageOutputStream when encoding the images.

ImageIO Cache Memory Threshold—Sets the threshold size (expressed in KiloBytes) which will make GeoServer choose between file cache vs memory based cache. If the estimated size of the image to be encoded is smaller than the threshold value, a *MemoryCacheImageOutputStream* will be used resulting into caching the image in memory. If the estimated size of the image to be encoded is greater than the threshold value, a *FileCacheImageOutputStream* will be used.

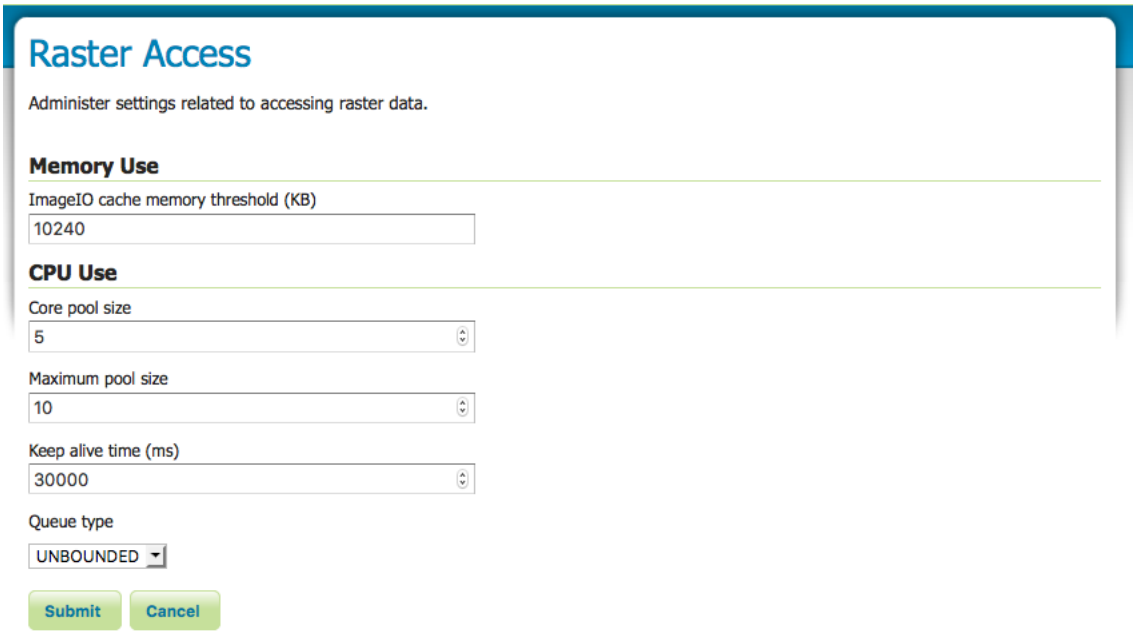


Fig. 9.9: Raster Access Settings

9.5.2 CPU Use

The imageMosaic reader may load, in parallel, different files that make up the mosaic by means of a ThreadPoolExecutor . A global ThreadPoolExecutor instance is shared by all the readers supporting and using concurrent reads. This section of the Coverage Access Settings administration page allows configuration of the Executor parameters.

Core Pool Size—Sets the core pool size of the thread pool executor. A positive integer must be specified.

Maximum Pool Size—Sets the maximum pool size of the thread pool executor. A positive integer must be specified.

Keep Alive Time—Sets the time to be wait by the executor before terminating an idle thread in case there are more threads than *corePoolSize*.

Queue Type—The executor service uses a BlockingQueue to manage submitted tasks. Using an *unbounded* queue is recommended which allows to queue all the pending requests with no limits (Unbounded). With a *direct* type, incoming requests will be rejected when there are already *maximumPoolSize* busy threads.

Note: If a new task is submitted to the list of tasks to be executed, and less than *corePoolSize* threads are running, a new thread is created to handle the request. Incoming tasks are queued in case *corePoolSize* or more threads are running.

Note: If a request can't be queued or there are less than *corePoolSize* threads running, a new thread is created unless this would exceed *maximumPoolSize*.

Note: If the pool currently has more than *corePoolSize* threads, excess threads will be terminated if they

have been idle for more than the *keepAliveTime*.

Note: If a new task is submitted to the list of tasks to be executed and there are more than *corePoolSize* but less than *maximumPoolSize* threads running, a new thread will be created only if the queue is full. This means that when using an *Unbounded* queue, no more threads than *corePoolSize* will be running and *keepAliveTime* has no influence.

Note: If *corePoolSize* and *maximumPoolSize* are the same, a fixed-size thread pool is used.

9.6 REST Configuration

Note: For more information, please see the section on *REST*.

The RESTful API allows to create new stores and append new granules to mosaics via file uploads. By default the new stores and granules are saved with the following directory structure:

```
$GEOSERVER_DATA_DIR/data/<workspace>/<store>[/<file>]
```

For changing the *Root Directory* from *$GEOSERVER_DATA_DIR/data* to another directory, the user can define a parameter called *Root Directory path* inside *Global Settings Page* and *Workspace Settings Page*.

In order to avoid cross workspace contamination, the final path will always be:

```
${rootDirectory}/workspace/store[/<file>]
```

Path remapping is achieved by using the default implementation of the **RESTUploadPathMapper** interface. This interface gives the possibility to also map the file position inside the store, which could be useful for harvesting files into an existing mosaic DataStore.

9.7 Advanced log configuration

GeoServer logging subsystem is based on Java logging, which is in turn by default redirected to Log4J and controlled by the current logging configuration set in the *Global Settings*.

The standard configuration can be overridden in a number of ways to create custom logging profiles or to force GeoServer to use another logging library altogheter.

9.7.1 Custom logging profiles

Anyone can write a new logging profile by adding a Log4J configuration file to the list of files already available in the `$GEOSERVER_DATA_DIR/logs` folder. The name of the file will become the configuration name displayed in the admin console and the contents will drive the specific behavior of the logger.

Here is an example, taken from the `GEOTOOLS_DEVELOPER_LOGGING` configuration, which enables the geotools log messages to appear in the logs:

```
log4j.rootLogger=WARN, geoserverlogfile, stdout

log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{dd MMM HH:mm:ss} %p [%c] - %m%n

log4j.category.log4j=FATAL

log4j.appender.geoserverlogfile=org.apache.log4j.RollingFileAppender
# Keep three backup files.
log4j.appender.geoserverlogfile.MaxBackupIndex=3
# Pattern to output: date priority [category] - message
log4j.appender.geoserverlogfile.layout=org.apache.log4j.PatternLayout
log4j.appender.geoserverlogfile.layout.ConversionPattern=%d %p [%c] - %m%n


log4j.category.org.geotools=TRACE
# Some more geotools loggers you may be interest in tweaking
log4j.category.org.geotools.factory=TRACE
log4j.category.org.geotools.renderer=DEBUG
log4j.category.org.geotools.data=TRACE
log4j.category.org.geotools.feature=TRACE
log4j.category.org.geotools.filter=TRACE
log4j.category.org.geotools.factory=TRACE

log4j.category.org.geoserver=INFO
log4j.category.org.vfny.geoserver=INFO

log4j.category.org.springframework=WARN
```

Any custom configuration can be setup to enable specific packages to emit logs at the desired logging level. There are however a few rules to follow:

- the configuration should always include a `geoserverlogfile` appender that GeoServer will configure to work against the location configured in the *Global Settings*
- a logger writing to the standard output should be called `stdout` and again GeoServer will enable/disable it according to the configuration set in the *Global Settings*
- it is advisable, but not require, to setup log rolling for the `geoserverlogfile` appender

9.7.2 Overriding the log location setup in the GeoServer configuration

When setting up a cluster of GeoServer machines it is common to share a single data directory among all the cluster nodes. There is however a gotcha, all nodes would end up writing the logs in the same file, which would cause various kinds of troubles depending on the operating system file locking rules (a single server might be able to write, or all togheter in an uncontrolled manner resulting in an unreadable log file).

In this case it is convenient to set a separate log location for each GeoServer node by setting the following parameter among the JVM system variables, enviroment variables, or servlet context parameters:

```
GEOSERVER_LOG_LOCATION=<the location of the file>
```

A common choice could be to use the machine name as a distinction, setting values such as `logs/geoserver_node1.log`, `logs/geoserver_node2.log` and so on: in this case all the log files would still be contained in the data directory and properly rotated, but each server would have its own separate log file to write on.

9.7.3 Forcing GeoServer to relinquish Log4J control

GeoServer internally overrides the Log4J configuration by using the current logging configuration as a template and appling the log location and standard output settings configured by the administrator.

If you wish GeoServer not to override the normal Log4J behavior you can set the following parameter among the JVM system variables, enviroment variables, or servlet context parameters:

```
RELINQUISH_LOG4J_CONTROL=true
```

9.7.4 Forcing GeoServer to use an alternate logging redirection

GeoServer uses the GeoTools logging framework, which in turn is based on Java Logging, but allowing to redirect all message to an alternate framework of users choice.

By default GeoServer setups a Log4J redirection, but it is possible to configure GeoServer to use plain Java Logging or Commons Logging instead (support for other loggers is also possible by using some extra programming).

If you wish to force GeoServer to use a different logging mechanism set the following parameters among the JVM system variables, enviroment variables, or servlet context parameters:

```
GT2_LOGGING_REDIRECTION=[JavaLogging,CommonsLogging,Log4J]
RELINQUISH_LOG4J_CONTROL=true
```

As noted in the example you'll also have to demand that GeoServer does not exert control over the Log4J configuration

9.8 Coordinate Reference System Handling

This section describes how coordinate reference systems (CRS) are handled in GeoServer, as well as what can be done to extend GeoServer's CRS handling abilities.

9.8.1 Coordinate Reference System Configuration

When adding data, GeoServer tries to inspect data headers looking for an EPSG code:

- If the data has a CRS with an explicit EPSG code and the full CRS definition behind the code matches the one in GeoServer, the CRS will be already set for the data.
- If the data has a CRS but no EPSG code, you can use the *Find* option on the *Layers* page to make GeoServer perform a lookup operation where the data CRS is compared against every other known CRS. If this succeeds, an EPSG code will be selected. The common case for a CRS that has no EPSG code is shapefiles whose .PRJ file contains a valid WKT string without the EPSG identifiers (as these are optional).

If an EPSG code cannot be found, then either the data has no CRS or it is unknown to GeoServer. In this case, there are a few options:

- Force the declared CRS, ignoring the native one. This is the best solution if the native CRS is known to be wrong.
- Reproject from the native to the declared CRS. This is the best solution if the native CRS is correct, but cannot be matched to an EPSG number. (An alternative is to add a custom EPSG code that matches exactly the native SRS. See the section on *Custom CRS Definitions* for more information.)

If your data has no native CRS information, the only option is to specify/force an EPSG code.

Increasing Comparison Tolerance

Decimal numbers comparisons are made using a comparison tolerance. This means, as an instance, that an ellipsoid's semi_major axis equals a candidate EPSG's ellipsoid semi_major axis only if their difference is within that tolerance. Default value is 10^-9 although it can be changed by setting a COMPARISON_TOLERANCE Java System property to your container's JVM to specify a different value.

Warning: The default value should be changed only if you are aware of use cases which require a wider tolerance. Don't change it unless really needed (See the following example).

Example

- Your sample dataset is known to be a LambertConformalConic projection and the related EPSG code defines latitude_of_origin value = 25.0.
- The coverageStore plugin is exposing raster projection details through a third party library which provides projection parameter definitions as float numbers.
- Due to the underlying math computations occurring in that third party library, the exposed projection parameters are subject to some accuracy loss, so that the provided latitude_of_origin is something like 25.0000012 whilst all the other params match the EPSG definition.
- You notice that the native CRS isn't properly recognized as the expected EPSG due to that small difference in latitude_of_origin

In that case you could consider increasing a bit the tolerance.

9.8.2 Custom CRS Definitions

Add a custom CRS

This example shows how to add a custom projection in GeoServer.

1. The projection parameters need to be provided as a WKT (well known text) definition. The code sample below is just an example:

```
PROJCS["NAD83 / Austin",
  GEOGCS["NAD83",
    DATUM["North_American_Datum_1983",
      SPHEROID["GRS 1980", 6378137.0, 298.257222101],
      TOWGS84[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]],
    PRIMEM["Greenwich", 0.0],
    UNIT["degree", 0.017453292519943295],
    AXIS["Lon", EAST],
    AXIS["Lat", NORTH]],
  PROJECTION["Lambert_Conformal_Conic_2SP"],
  PARAMETER["central_meridian", -100.333333333333],
  PARAMETER["latitude_of_origin", 29.6666666666667],
  PARAMETER["standard_parallel_1", 31.883333333333297],
  PARAMETER["false_easting", 2296583.333333],
  PARAMETER["false_northing", 9842500.0],
```

```
  PARAMETER["standard_parallel_2", 30.1166666666667],
  UNIT["m", 1.0],
  AXIS["x", EAST],
  AXIS["y", NORTH],
  AUTHORITY["EPSG","100002"]]
```

Note: This code sample has been formatted for readability. The information will need to be provided on a single line instead, or with backslash characters at the end of every line (except the last one).

2. Go into the `user_projections` directory inside your data directory, and open the `epsg.properties` file. If this file doesn't exist, you can create it.
3. Insert the code WKT for the projection at the end of the file (on a single line or with backslash characters):

```
100002=PROJCS["NAD83 / Austin", \
  GEOGCS["NAD83", \
    DATUM["North_American_Datum_1983", \
      SPHEROID["GRS 1980", 6378137.0, 298.257222101], \
      TOWGS84[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]], \
    PRIMEM["Greenwich", 0.0], \
    UNIT["degree", 0.017453292519943295], \
    AXIS["Lon", EAST], \
    AXIS["Lat", NORTH]], \
  PROJECTION["Lambert_Conformal_Conic_2SP"], \
  PARAMETER["central_meridian", -100.333333333333], \
  PARAMETER["latitude_of_origin", 29.6666666666667], \
  PARAMETER["standard_parallel_1", 31.883333333333297], \
  PARAMETER["false_easting", 2296583.333333], \
  PARAMETER["false_northing", 9842500.0], \
  PARAMETER["standard_parallel_2", 30.1166666666667], \
  UNIT["m", 1.0], \
  AXIS["x", EAST], \
  AXIS["y", NORTH], \
  AUTHORITY["EPSG","100002"]]
```

Note: Note the number that precedes the WKT. This will determine the EPSG code. So in this example, the EPSG code is 100002.

1. Save the file.
2. Restart GeoServer.
3. Verify that the CRS has been properly parsed by navigating to the *SRS List* page in the *Web administration interface*.
4. If the projection wasn't listed, examine the logs for any errors.

Override an official EPSG code

In some situations it is necessary to override an official EPSG code with a custom definition. A common case is the need to change the TOWGS84 parameters in order to get better reprojection accuracy in specific areas.

The GeoServer referencing subsystem checks the existence of another property file, `epsg_overrides.properties`, whose format is the same as `epsg.properties`. Any definition contained in `epsg_overrides.properties` will **override** the EPSG code, while definitions stored in `epsg.proeprties` can only **add** to the database.

Special care must be taken when overriding the Datum parameters, in particular the **TOWGS84** parameters. To make sure the override parameters are actually used the code of the Datum must be removed, otherwise the referencing subsystem will keep on reading the official database in search of the best Datum shift method (grid, 7 or 5 parameters transformation, plain affine transform).

For example, if you need to override the official **TOWGS84** parameters of EPSG:23031:

```
PROJCS["ED50 / UTM zone 31N",
  GEOGCS["ED50",
    DATUM["European Datum 1950",
      SPHEROID["International 1924", 6378388.0, 297.0, AUTHORITY["EPSG","7022"]],
      TOWGS84[-157.89, -17.16, -78.41, 2.118, 2.697, -1.434, -1.1097046576093785],
      AUTHORITY["EPSG","6230"]],
    PRIMEM["Greenwich", 0.0, AUTHORITY["EPSG","8901"]],
    UNIT["degree", 0.017453292519943295],
    AXIS["Geodetic longitude", EAST],
    AXIS["Geodetic latitude", NORTH],
    AUTHORITY["EPSG","4230"]],
 PROJECTION["Transverse_Mercator"],
  PARAMETER["central_meridian", 3.0],
  PARAMETER["latitude_of_origin", 0.0],
  PARAMETER["scale_factor", 0.9996],
  PARAMETER["false_easting", 500000.0],
  PARAMETER["false_northing", 0.0],
  UNIT["m", 1.0],
  AXIS["Easting", EAST],
  AXIS["Northing", NORTH],
  AUTHORITY["EPSG","23031"]]
```

You should write the following (in a single line, here it's reported formatted over multiple lines for readability):

```
23031=
 PROJCS["ED50 / UTM zone 31N",
   GEOGCS["ED50",
     DATUM["European Datum 1950",
       SPHEROID["International 1924", 6378388.0, 297.0, AUTHORITY["EPSG","7022"]],
       TOWGS84[-136.65549, -141.4658, -167.29848, 2.093088, 0.001405, 0.107709, 11.
↪54611],
       AUTHORITY["EPSG","6230"]],
     PRIMEM["Greenwich", 0.0, AUTHORITY["EPSG","8901"]],
     UNIT["degree", 0.017453292519943295],
     AXIS["Geodetic longitude", EAST],
     AXIS["Geodetic latitude", NORTH]],
  PROJECTION["Transverse_Mercator"],
   PARAMETER["central_meridian", 3.0],
   PARAMETER["latitude_of_origin", 0.0],
   PARAMETER["scale_factor", 0.9996],
   PARAMETER["false_easting", 500000.0],
   PARAMETER["false_northing", 0.0],
   UNIT["m", 1.0],
   AXIS["Easting", EAST],
   AXIS["Northing", NORTH],
   AUTHORITY["EPSG","23031"]]
```

The definition has been changed in two places, the **TOWGS84** paramerers, and the Datum code, `AUTHORITY["EPSG","4230"]`, has been removed.

9.8.3 Coordinate Operations

Coordinate operations are used to convert coordinates from a *source CRS* to a *target CRS*.

If source and target CRSs are refered to a different datum, a datum transform has to be applied. Datum transforms are not exact, they are determined empirically. For the same pair of CRS, there can be many datum transforms and versions, each one with its own domain of validity and an associated transform error. Given a CRS pair, GeoServer will automatically pick the most accurate datum transform from the EPSG database, unless a custom operation is declared.

- Coordinate operations can be queried and tested using the *Reprojection Console*.
- To enable higher accuracy Grid Shift transforms, see *Add Grid Shift Transform files*.
- See *Define a custom Coordinate Operation* to declare new operations. Custom operations will take precedence over the EPSG ones.

Reprojection Console

The reprojection console (available in *Demos*) lets you quickly test coordinate operations. Use it to convert a single coordinate or WKT geometry, and to see the operation details GeoServer is using. It is also useful to learn by example when you have to *Define a custom Coordinate Operation*.

Read more about the *Reprojection console*.

Add Grid Shift Transform files

GeoServer supports NTv2 and NADCON grid shift transforms. Grid files are not shipped out with GeoServer. They need to be downloaded, usually from yor National Mapping Agency website.

Warning: Grid Shift files are only valid in the specific geographic domain for which they where made; trying to transform coordinates outside this domain will result in no trasformation at all. Make sure that the Grid Shift files are valid in the area you want to transform.

1. Search for the *Grid File Name(s)* int the tables below, which are extracted from EPSG version 7.9.0. If you need to use a Grid Shift transform not declared in EPSG, you will need to *Define a custom Coordinate Operation*.
2. Get the Grid File(s) from your National Mapping Agency (NTv2) or the US National Geodetic Survey (NADCON).
3. Copy the Grid File(s) in the `user_projections` directory inside your data directory.
4. Use the *Reprojection Console* to test the new transform.

List of available Grid Shift transforms

The list of Grid Shift transforms declared in EPSG version 7.9.0 is:

NTv2

Source CRS	Target CRS	Grid File Name	Source Info
4122	4326	NB7783v2.gsb	OGP
4122	4326	NS778301.gsb	OGP
4122	4326	PE7783V2.gsb	OGP
4122	4617	NB7783v2.gsb	New Brunswick Geographic Information Corporatio
4122	4617	NS778301.gsb	Nova Scotia Geomatics Centre - Contact aflemmin@l
4122	4617	PE7783V2.gsb	PEI Department of Transportation & Public Works
4149	4150	CHENYX06.gsb	Bundesamt für Landestopographie; www.swisstopo.
4171	4275	rgf93_ntf.gsb	ESRI
4202	4283	A66 National (13.09.01).gsb	GDA Technical Manual. http://www.icsm.gov.au/g
4202	4283	SEAust_21_06_00.gsb	Office of Surveyor General Victoria; http://www.lan
4202	4283	nt_0599.gsb	GDA Technical Manual. http://www.icsm.gov.au/g
4202	4283	tas_1098.gsb	http://www.delm.tas.gov.au/osg/Geodetic_transfo
4202	4283	vic_0799.gsb	Office of Surveyor General Victoria; http://www.lan
4202	4326	A66 National (13.09.01).gsb	OGP
4203	4283	National 84 (02.07.01).gsb	GDA Technical Manual. http://www.icsm.gov.au/g
4203	4283	wa_0400.gsb	http://www.dola.wa.gov.au/lotl/survey_geodesy/g
4203	4283	wa_0700.gsb	Department of Land Information, Government of We
4203	4326	National 84 (02.07.01).gsb	OGP
4225	4326	CA7072_003.gsb	OGP
4225	4674	CA7072_003.gsb	IBGE.
4230	4258	SPED2ETV2.gsb	Instituto Geográfico Nacional, www.cnig.es
4230	4258	sped2et.gsb	Instituto Geográfico Nacional, www.cnig.es
4230	4326	SPED2ETV2.gsb	OGP
4230	4326	sped2et.gsb	OGP
4258	4275	rgf93_ntf.gsb	OGP
4267	4269	NTv2_0.gsb	https://open.canada.ca/data/en/dataset/b3534942-
4267	4269	QUE27-83.gsb	Geodetic Service of Quebec. Contact alain.bernard@r
4267	4326	NTv2_0.gsb	OGP
4267	4326	QUE27-98.gsb	OGP
4267	4326	SK27-98.gsb	OGP
4267	4617	QUE27-98.gsb	Geodetic Service of Quebec. Contact alain.bernard@r
4267	4617	SK27-98.gsb	Dir Geodetic Surveys; SaskGeomatics Div.; Saskatche
4269	4326	AB_CSRS.DAC	OGP
4269	4326	NAD83-98.gsb	OGP
4269	4326	SK83-98.gsb	OGP
4269	4617	AB_CSRS.DAC	Geodetic Control Section; Land and Forest Svc; Alber
4269	4617	NAD83-98.gsb	Geodetic Service of Quebec. Contact alain.bernard@r
4269	4617	SK83-98.gsb	Dir Geodetic Surveys; SaskGeomatics Div.; Saskatche
4272	4167	nzgd2kgrid0005.gsb	Land Information New Zealand: LINZS25000 Standa
4272	4326	nzgd2kgrid0005.gsb	OGP
4277	4258	OSTN02_NTv2.gsb	Ordnance Survey of Great Britain, http://www.gps.g
4277	4326	OSTN02_NTv2.gsb	OGP
4314	4258	BETA2007.gsb	BKG via EuroGeographics http://crs.bkg.bund.de/c
4314	4326	BETA2007.gsb	OGP
4326	4275	rgf93_ntf.gsb	OGP
4608	4269	May76v20.gsb	Geodetic Survey of Canada Natural Resources Canad
4608	4326	May76v20.gsb	OGP

Table 9.1 – continued from previous page

Source CRS	Target CRS	Grid File Name	Source Info
4609	4269	CGQ77-83.gsb	Geodetic Service of Quebec. Contact alain.bernard@r
4609	4326	CGQ77-98.gsb	OGP
4609	4617	CGQ77-98.gsb	Geodetic Service of Quebec. Contact alain.bernard@r
4618	4326	SAD69_003.gsb	OGP
4618	4674	SAD69_003.gsb	IBGE.
4745	4326	BETA2007.gsb	OGP
4746	4326	BETA2007.gsb	OGP
4749	4644	RGNC1991_NEA74Noumea.gsb	ESRI
4749	4662	RGNC1991_IGN72GrandeTerre.gsb	ESRI
5524	4326	CA61_003.gsb	OGP
5524	4674	CA61_003.gsb	IBGE.
5527	4326	SAD96_003.gsb	OGP
5527	4674	SAD96_003.gsb	IBGE.

NADCON

Source CRS	Target CRS	Version	Latitude shift file	Longitude shift file
4135	4269	NGS-Usa HI	hawaii.las	hawaii.los
4136	4269	NGS-Usa AK StL	stlrnc.las	stlrnc.los
4137	4269	NGS-Usa AK StP	stpaul.las	stpaul.los
4138	4269	NGS-Usa AK StG	stgeorge.las	stgeorge.los
4139	4269	NGS-PRVI	prvi.las	prvi.los
4169	4152	NGS-Asm E	eshpgn.las	eshpgn.los
4169	4152	NGS-Asm W	wshpgn.las	wshpgn.los
4267	4269	NGS-Usa AK	alaska.las	alaska.los
4267	4269	NGS-Usa Conus	conus.las	conus.los
4269	4152	NGS-Usa AL	alhpgn.las	alhpgn.los
4269	4152	NGS-Usa AR	arhpgn.las	arhpgn.los
4269	4152	NGS-Usa AZ	azhpgn.las	azhpgn.los
4269	4152	NGS-Usa CA n	cnhpgn.las	cnhpgn.los
4269	4152	NGS-Usa CO	cohpgn.las	cohpgn.los
4269	4152	NGS-Usa CA s	cshpgn.las	cshpgn.los
4269	4152	NGS-Usa ID MT e	emhpgn.las	emhpgn.los
4269	4152	NGS-Usa TX e	ethpgn.las	ethpgn.los
4269	4152	NGS-Usa FL	flhpgn.las	flhpgn.los
4269	4152	NGS-Usa GA	gahpgn.las	gahpgn.los
4269	4152	NGS-Usa HI	hihpgn.las	hihpgn.los
4269	4152	NGS-Usa IA	iahpgn.las	iahpgn.los
4269	4152	NGS-Usa IL	ilhpgn.las	ilhpgn.los
4269	4152	NGS-Usa IN	inhpgn.las	inhpgn.los
4269	4152	NGS-Usa KS	kshpgn.las	kshpgn.los
4269	4152	NGS-Usa KY	kyhpgn.las	kyhpgn.los
4269	4152	NGS-Usa LA	lahpgn.las	lahpgn.los
4269	4152	NGS-Usa DE MD	mdhpgn.las	mdhpgn.los
4269	4152	NGS-Usa ME	mehpgn.las	mehpgn.los
4269	4152	NGS-Usa MI	mihpgn.las	mihpgn.los
4269	4152	NGS-Usa MN	mnhpgn.las	mnhpgn.los
4269	4152	NGS-Usa MO	mohpgn.las	mohpgn.los

Continued on next page

Table 9.2 – continued from previous page

Source CRS	Target CRS	Version	Latitude shift file	Longitude shift file
4269	4152	NGS-Usa MS	mshpgn.las	mshpgn.los
4269	4152	NGS-Usa NE	nbhpgn.las	nbhpgn.los
4269	4152	NGS-Usa NC	nchpgn.las	nchpgn.los
4269	4152	NGS-Usa ND	ndhpgn.las	ndhpgn.los
4269	4152	NGS-Usa NewEng	nehpgn.las	nehpgn.los
4269	4152	NGS-Usa NJ	njhpgn.las	njhpgn.los
4269	4152	NGS-Usa NM	nmhpgn.las	nmhpgn.los
4269	4152	NGS-Usa NV	nvhpgn.las	nvhpgn.los
4269	4152	NGS-Usa NY	nyhpgn.las	nyhpgn.los
4269	4152	NGS-Usa OH	ohhpgn.las	ohhpgn.los
4269	4152	NGS-Usa OK	okhpgn.las	okhpgn.los
4269	4152	NGS-Usa PA	pahpgn.las	pahpgn.los
4269	4152	NGS-PRVI	pvhpgn.las	pvhpgn.los
4269	4152	NGS-Usa SC	schpgn.las	schpgn.los
4269	4152	NGS-Usa SD	sdhpgn.las	sdhpgn.los
4269	4152	NGS-Usa TN	tnhpgn.las	tnhpgn.los
4269	4152	NGS-Usa UT	uthpgn.las	uthpgn.los
4269	4152	NGS-Usa VA	vahpgn.las	vahpgn.los
4269	4152	NGS-Usa WI	wihpgn.las	wihpgn.los
4269	4152	NGS-Usa ID MT w	wmhpgn.las	wmhpgn.los
4269	4152	NGS-Usa OR WA	wohpgn.las	wohpgn.los
4269	4152	NGS-Usa TX w	wthpgn.las	wthpgn.los
4269	4152	NGS-Usa WV	wvhpgn.las	wvhpgn.los
4269	4152	NGS-Usa WY	wyhpgn.las	wyhpgn.los
4675	4152	NGS-Gum	guhpgn.las	guhpgn.los

Define a custom Coordinate Operation

Custom Coordinate Operations are defined in `epsg_operations.properties` file. This file has to be placed into the `user_projections` directory, inside your data directory (create it if it doesn't exist).

Each line in `epsg_operations.properties` will describe a coordinate operation consisting of a *source CRS*, a *target CRS*, and a math transform with its parameter values. Use the following syntax:

```
<source crs code>,<target crs code>=<WKT math transform>
```

Math transform is described in Well-Known Text syntax. Parameter names and value ranges are described in the EPSG Geodetic Parameter Registry.

Note: Use the *Reprojection Console* to learn from example and to test your custom definitions.

Examples

Custom NTv2 file:

```
4230,4258=PARAM_MT["NTv2", \
  PARAMETER["Latitude and longitude difference file", "100800401.gsb"]]
```

Geocentric transformation, preceded by an ellipsoid to geocentric conversion, and back geocentric to ellipsoid. The results is a concatenation of three math transforms:

```
4230,4258=CONCAT_MT[ \
  PARAM_MT["Ellipsoid_To_Geocentric", \
    PARAMETER["dim", 2], \
    PARAMETER["semi_major", 6378388.0], \
    PARAMETER["semi_minor", 6356911.9461279465]], \
  PARAM_MT["Position Vector transformation (geog2D domain)", \
    PARAMETER["dx", -116.641], \
    PARAMETER["dy", -56.931], \
    PARAMETER["dz", -110.559], \
    PARAMETER["ex", 0.8925078166311858], \
    PARAMETER["ey", 0.9207660950870382], \
    PARAMETER["ez", -0.9166407989620964], \
    PARAMETER["ppm", -3.5200000000346066]], \
  PARAM_MT["Geocentric_To_Ellipsoid", \
    PARAMETER["dim", 2], \
    PARAMETER["semi_major", 6378137.0], \
    PARAMETER["semi_minor", 6356752.314140356]]]
```

You can make use of existing grid shift files such as this explicit transformation from NAD27 to WGS84 made up of a NADCON transform from NAD27 to NAD83 followed by a Molodenski transform converting from the GRS80 Ellipsoid (used by NAD83) to the WGS84 Ellipsoid:

```
4267,4326=CONCAT_MT[ \
  PARAM_MT["NADCON", \
    PARAMETER["Latitude difference file", "conus.las"], \
    PARAMETER["Longitude difference file", "conus.los"]], \
  PARAM_MT["Molodenski", \
    PARAMETER["dim", 2], \
    PARAMETER["dx", 0.0], \
    PARAMETER["dy", 0.0], \
    PARAMETER["dz", 0.0], \
    PARAMETER["src_semi_major", 6378137.0], \
    PARAMETER["src_semi_minor", 6356752.314140356], \
    PARAMETER["tgt_semi_major", 6378137.0], \
    PARAMETER["tgt_semi_minor", 6356752.314245179]]]
```

Affine 2D transform operating directly in projected coordinates:

```
23031,25831=PARAM_MT["Affine", \
  PARAMETER["num_row", 3], \
  PARAMETER["num_col", 3], \
  PARAMETER["elt_0_0", 1.0000015503712145], \
  PARAMETER["elt_0_1", 0.00000758753979846734], \
  PARAMETER["elt_0_2", -129.549], \
  PARAMETER["elt_1_0", -0.00000758753979846734], \
  PARAMETER["elt_1_1", 1.0000015503712145], \
  PARAMETER["elt_1_2", -208.185]]
```

Each operation can be described in a single line, or can be split in several lines for readability, adding a backslash "" at the end of each line, as in the former examples.

9.8.4 Manually editing the EPSG database

Warning: These instructions are very advanced, and are here mainly for the curious who want to know details about the EPSG database subsystem.

To define a custom projection, edit the EPSG.sql file, which is used to create the cached EPSG database.

1. Navigate to the `WEB-INF/lib` directory
2. Uncompress the `gt2-epsg-h.jar` file. On Linux, the command is:

```
jar xvf gt2-epsg-h.jar
```

3. Open `org/geotools/referencing/factory/epsg/EPSG.sql` with a text editor. To add a custom projection, these entries are essential:
   (a) An entry in the EPSG_COORDINATEREFERENCESYSTEM table:

```
(41111,'WGC 84 / WRF Lambert',1324,'projected',4400,NULL,4326,20000,NULL,NULL,
↪'US Nat. scale mapping.','Entered by Alex Petkov','Missoula Firelab WRF',
↪'WRF','2000-10-19','',1,0),
```

   where:

   - **1324** is the EPSG_AREA code that describes the area covered by my projection
   - **4400** is the EPSG_COORDINATESYSTEM code for my projection
   - **20000** is the EPSG_COORDOPERATIONPARAMVALUE key for the array that contains my projection parameters

   (b) An entry in the EPSG_COORDOPERATIONPARAMVALUE table:

```
(20000,9802,8821,40,'',9102),    //latitude of origin
(20000,9802,8822,-97.0,'',9102), //central meridian
(20000,9802,8823,33,'',9110),    //st parallel 1
(20000,9802,8824,45,'',9110),    //st parallel 2
(20000,9802,8826,0.0,'',9001),   //false easting
(20000,9802,8827,0.0,'',9001)    //false northing
```

   where:

   - **9802** is the EPSG_COORDOPERATIONMETHOD key for the Lambert Conic Conformal (2SP) formula

   (c) An entry in the EPSG_COORDOPERATION table:

   (20000,'WRF Lambert','conversion',NULL,NULL,'',NULL,1324,'Used for weather forecasting.',0.0,9802,NULL,NULL,'Used with the WRF-Chem model for weather forecasting','Firelab in Missoula, MT','EPSG','2005-11-23','2005.01',1,0)

   where:

   - **1324** is the EPSG_AREA code that describes the area covered by my projection
   - **9802** is the EPSG_COORDOPERATIONMETHOD key for the Lambert Conic Conformal (2SP) formula

Note: Observe the commas. If you enter a line that is at the end of an INSERT statement, the comma is omitted (make sure the row before that has a comma at the end). Otherwise, add a comma at the end of your entry.

1. After all edits, save the file and exit.
2. Compress the gt2-epsg-h.jar file. On Linux, the command is:

```
jar -Mcvf gt2-epsg-h.jar META-INF org
```

3. Remove the cached copy of the EPSG database, so that can be recreated. On Linux, the command is:

```
rm -rf /tmp/Geotools/Databases/HSQL
```

4. Restart GeoServer.

The new projection will be successfully parsed. Verify that the CRS has been properly parsed by navigating to the *SRS List* page in the *Web administration interface*.

9.9 Virtual Services

The different types of services in GeoServer include WFS, WMS, and WCS, commonly referred to as "OWS" services. These services are global in that each service publishes every layer configured on the server. WFS publishes all vector layer (feature types), WCS publishes all raster layers (coverages), and WMS publishes everything.

A *virtual service* is a view of the global service that consists only of a subset of the layers. Virtual services are based on GeoServer workspaces. For each workspace that exists a virtual service exists along with it. The virtual service publishes only those layers that fall under the corresponding workspace.

Warning: Virtual services only apply to the core OWS services, and not OWS services accessed through GeoWebCache. It also does not apply to other subsystems such as REST.

When a client accesses a virtual service that client only has access to those layers published by that virtual service. Access to layers in the global service via the virtual service will result in an exception. This makes virtual services ideal for compartmentalizing access to layers. A service provider may wish to create multiple services for different clients handing one service url to one client, and a different service url to another client. Virtual services allow the service provider to achieve this with a single GeoServer instance.

9.9.1 Filtering by workspace

Consider the following snippets of the WFS capabilities document from the GeoServer release configuration that list all the feature types:

```
http://localhost:8080/geoserver/wfs?request=GetCapabilities

<wfs:WFS_Capabilities>

  <FeatureType xmlns:tiger="http://www.census.gov">
   <Name>tiger:poly_landmarks</Name>
--
```

```
  <FeatureType xmlns:tiger="http://www.census.gov">
   <Name>tiger:poi</Name>
--
  <FeatureType xmlns:tiger="http://www.census.gov">
   <Name>tiger:tiger_roads</Name>
--
  <FeatureType xmlns:sf="http://www.openplans.org/spearfish">
   <Name>sf:archsites</Name>
--
  <FeatureType xmlns:sf="http://www.openplans.org/spearfish">
   <Name>sf:bugsites</Name>
--
  <FeatureType xmlns:sf="http://www.openplans.org/spearfish">
   <Name>sf:restricted</Name>
--
  <FeatureType xmlns:sf="http://www.openplans.org/spearfish">
   <Name>sf:roads</Name>
--
  <FeatureType xmlns:sf="http://www.openplans.org/spearfish">
   <Name>sf:streams</Name>
--
  <FeatureType xmlns:topp="http://www.openplans.org/topp">
   <Name>topp:tasmania_cities</Name>
--
  <FeatureType xmlns:topp="http://www.openplans.org/topp">
   <Name>topp:tasmania_roads</Name>
--
  <FeatureType xmlns:topp="http://www.openplans.org/topp">
   <Name>topp:tasmania_state_boundaries</Name>
--
  <FeatureType xmlns:topp="http://www.openplans.org/topp">
   <Name>topp:tasmania_water_bodies</Name>
--
  <FeatureType xmlns:topp="http://www.openplans.org/topp">
   <Name>topp:states</Name>
--
  <FeatureType xmlns:tiger="http://www.census.gov">
   <Name>tiger:giant_polygon</Name>

</wfs:WFS_Capabilities>
```

The above document lists every feature type configured on the server. Now consider the following capabilities request:

```
http://localhost:8080/geoserver/topp/wfs?request=GetCapabilities
```

The part of interest in the above request is the "topp" prefix to the wfs service. The above url results in the following feature types in the capabilities document:

```
<wfs:WFS_Capabilities>

   <FeatureType xmlns:topp="http://www.openplans.org/topp">
    <Name>topp:tasmania_cities</Name>
 --
   <FeatureType xmlns:topp="http://www.openplans.org/topp">
    <Name>topp:tasmania_roads</Name>
 --
   <FeatureType xmlns:topp="http://www.openplans.org/topp">
```

```
    <Name>topp:tasmania_state_boundaries</Name>
--
  <FeatureType xmlns:topp="http://www.openplans.org/topp">
    <Name>topp:tasmania_water_bodies</Name>
--
  <FeatureType xmlns:topp="http://www.openplans.org/topp">
    <Name>topp:states</Name>

</wfs:WFS_Capabilities>
```

The above feature types correspond to those configured on the server as part of the "topp" workspace.

The consequence of a virtual service is not only limited to the capabilities document of the service. When a client accesses a virtual service it is restricted to only those layers for all operations. For instance, consider the following WFS feature request:

```
http://localhost:8080/geoserver/topp/wfs?request=GetFeature&typename=tiger:roads
```

The above request results in an exception. Since the request feature type "tiger:roads" is not in the "topp" workspace the client will receive an error stating that the requested feature type does not exist.

9.9.2 Filtering by layer

It is possible to further filter a global service by specifying the name of layer as part of the virtual service. For instance consider the following capabilities document:

```
http://localhost:8080/geoserver/topp/states/wfs?request=GetCapabilities
```

The part of interest is the "states" prefix to the wfs service. The above url results in the following capabilities document that contains a single feature type:

```
<wfs:WFS_Capabilities>

  <FeatureType xmlns:topp="http://www.openplans.org/topp">
    <Name>topp:states</Name>

<wfs:WFS_Capabilities>
```

9.9.3 Turning off global services

It is possible to completely restrict access to the global OWS services by setting a configuration flag. When global access is disabled OWS services may only occur through a virtual service. Any client that tries to access a service globally will receive an exception.

To disable global services log into the GeoServer web administration interface and navigate to "Global Settings". Uncheck the "Enable Global Services" check box.

9.10 Demos

This page contains helpful links to various information pages regarding GeoServer and its features. You do not need to be logged into GeoServer to access this page.

The page contains the following options

- *Demo Requests*
- *SRS List*
- *Reprojection console*
- *WCS Request Builder*

Fig. 9.10: Demos page

If you have the *WPS* extension installed, you will see an additional option:

- *WPS Request Builder*

Fig. 9.11: Demos page with WPS extension installed

9.10.1 Demo Requests

This page has example WMS, WCS, and WFS requests for GeoServer that you can use, examine, and change. Select a request from the drop down list.

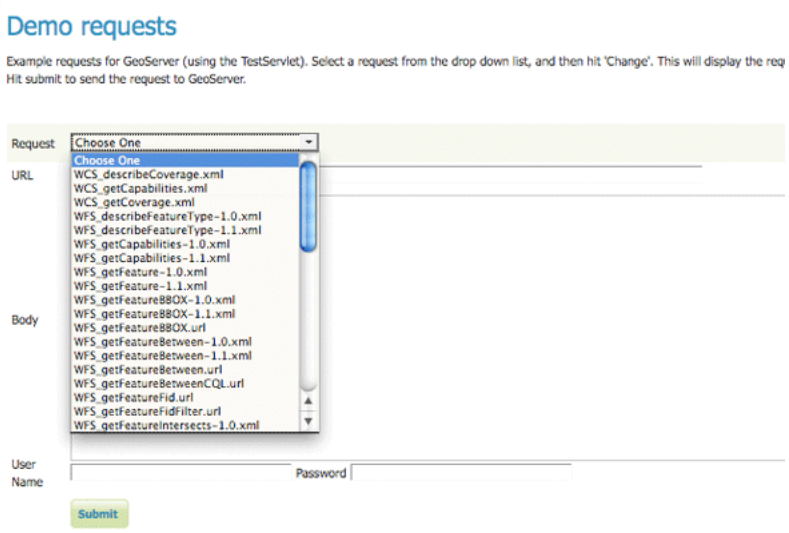


Fig. 9.12: Selecting demo requests

Both *Web Feature Service (WFS)* as well as *Web Coverage Service (WCS)* requests will display the request URL and the XML body. *Web Map Service (WMS)* requests will only display the request URL.

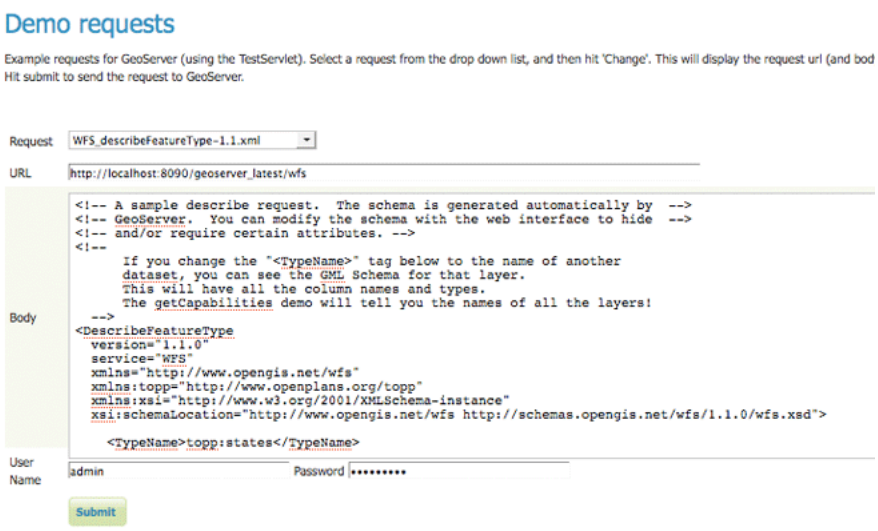


Fig. 9.13: WFS 1.1 DescribeFeatureType sample request

Click *Submit* to send the request to GeoServer. For WFS and WCS requests, GeoServer will automatically generate an XML reponse.

Submitting a WMS GetMap request displays an image based on the provided geographic data.

WMS GetFeatureInfo requests retrieve information regarding a particular feature on the map image.

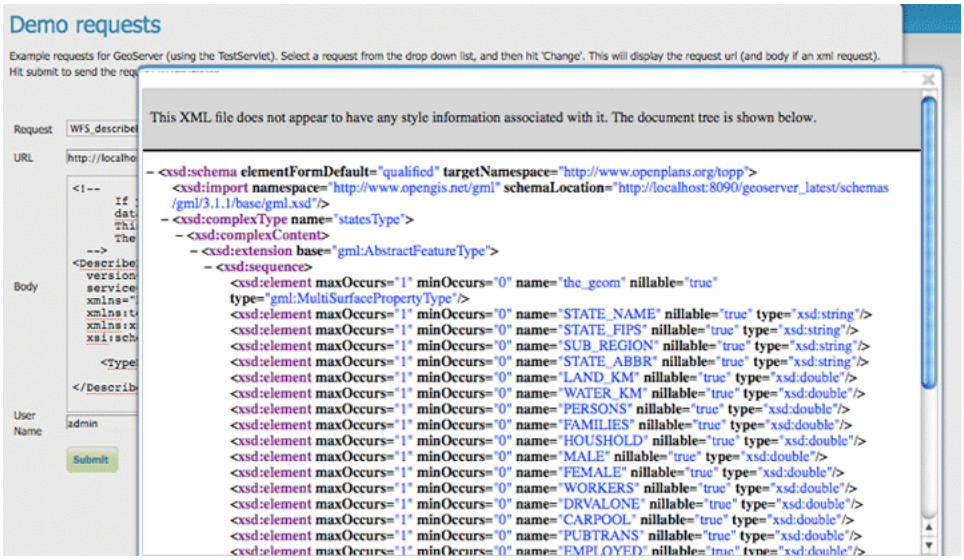


Fig. 9.14: XML response from a WFS 1.1 DescribeFeatureType sample request

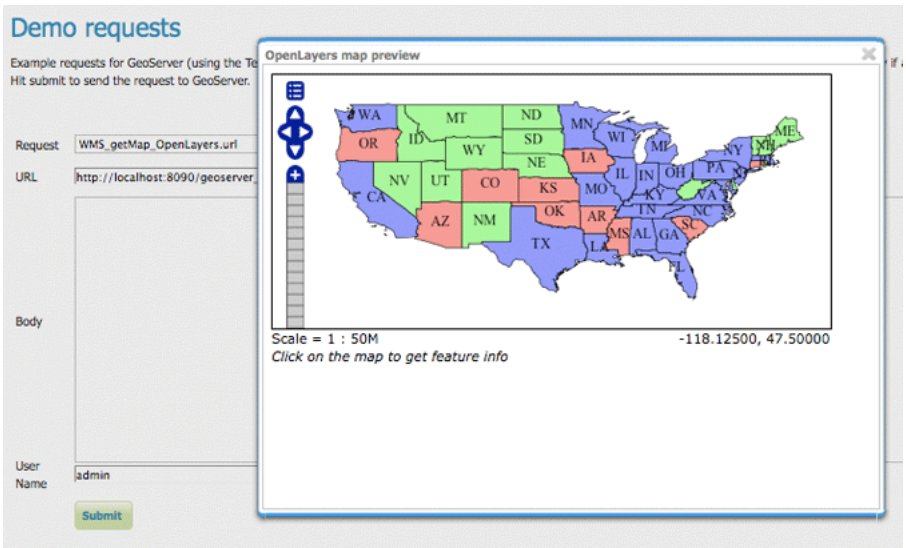


Fig. 9.15: OpenLayers WMS GetMap request

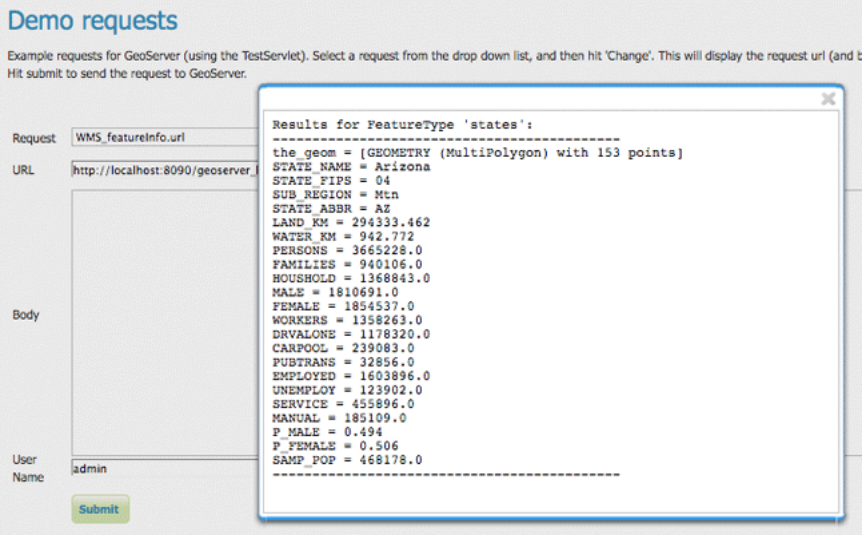


Fig. 9.16: WMS GetFeatureInfo request

9.10.2 SRS List

GeoServer natively supports almost 4,000 Spatial Referencing Systems (SRS), also known as **projections**, and more can be added. A spatial reference system defines an ellipsoid, a datum using that ellipsoid, and either a geocentric, geographic or projection coordinate system. This page lists all SRS info known to GeoServer.

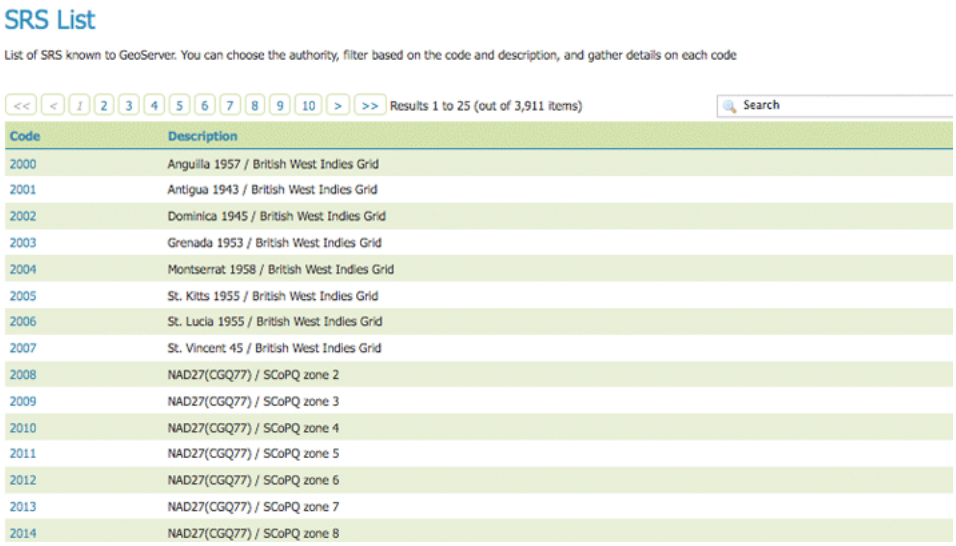


Fig. 9.17: Listing of all Spatial Referencing Systems (SRS) known to GeoServer

The *Code* column refers to the unique integer identifier defined by the author of that spatial reference system. Each code is linked to a more detailed description page, accessed by clicking on that code.

The title of each SRS is composed of the author name and the unique integer identifier (code) defined by the Author. In the above example, the author is the European Petroleum Survey Group (EPSG) and the Code is 2000. The fields are as follows:

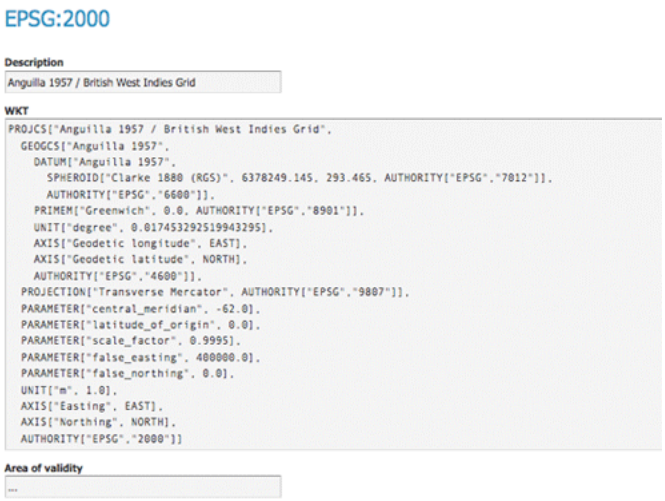


Fig. 9.18: Details for SRS EPSG:2000

Description—A short text description of the SRS

WKT—A string describing the SRS. WKT stands for "Well Known Text"

Area of Validity—The bounding box for the SRS

9.10.3 Reprojection console

The reprojection console allows you to calculate and test coordinate transformation. You can input a single coordinate or WKT geometry, and transform it from one CRS to another.

For example, you can use the reprojection console to transform a bounding box (as a WKT polygon or line) between different CRSs.

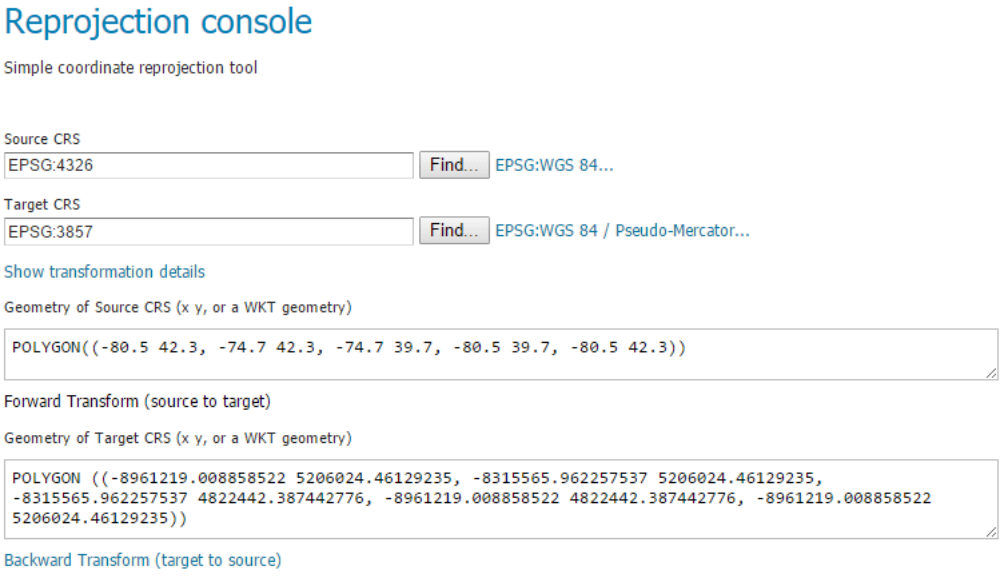


Fig. 9.19: Reprojection console showing a transformed bounding box

Use *Forward transformation* to convert from source CRS to target CRS, and *Backward transformation* to convert from target CRS to source CRS.

You can also view the underlying calculation GeoServer is using to perform the transformation.

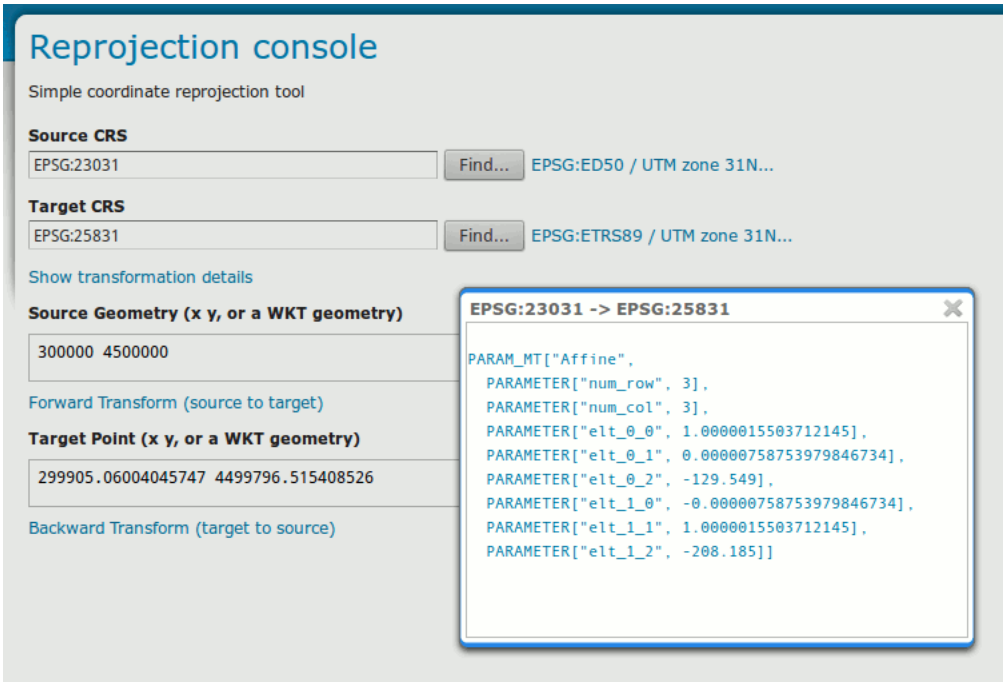


Fig. 9.20: Reprojection console showing operation details

Read more about *Coordinate Reference System Handling*.

9.10.4 WCS Request Builder

The WCS Request Builder is a tool for generating and executing WCS requests. Since WCS requests can be cumbersome to author, this tool can make working with WCS much easier.

Read more about the *WCS Request Builder*.

9.10.5 WPS Request Builder

GeoServer with the *WPS extension installed* includes a request builder for generating and executing WPS processes. Since WPS requests can be cumbersome to author, this tool can make working with WPS much easier.

Read more about the *WPS Request Builder*.

CHAPTER 10

GeoServer data directory

The GeoServer **data directory** is the location in the file system where GeoServer stores its configuration information.

The configuration defines what data is served by GeoServer, where it is stored, and how services interact with and serve the data. The data directory also contains a number of support files used by GeoServer for various purposes.

For production use, it is recommended to define an *external* data directory (outside the application) to make it easier to upgrade. The *Setting the data directory location* section describes how to configure GeoServer to use an existing data directory.

Note: Since GeoServer provides both an interactive interface (via the *web admin interface*) and programmatic interface (through the *REST API*) to manage configuration, most users do not need to know about the *internal structure of the data directory*, but an overview is provided below.

10.1 Data directory default location

If GeoServer is running in **standalone** mode (via an installer or a binary) the data directory is located at `<installation root>/data_dir`.

Standalone platform	Default/typical location
Windows (except XP)	`C:\Program Files (x86)\GeoServer 2.15.1\data_dir`
Windows XP	`C:Program Files\GeoServer 2.15.1\data_dir`
Mac OS X	`/Applications/GeoServer.app/Contents/Resources/Java/data_dir`
Linux (Tomcat)	`/var/lib/tomcat7/webapps/geoserver/data`

If GeoServer is running as a **web archive** inside of a custom-deployed application server, the data directory is by default located at `<web application root>/data`.

10.1.1 Creating a new data directory

The easiest way to create a new data directory is to copy an existing one.

Once the data directory has been located, copy it to a new location. To point a GeoServer instance at the new data directory proceed to the next section *Setting the data directory location*.

10.2 Setting the data directory location

The procedure for setting the location of the GeoServer data directory is dependent on the type of GeoServer installation. Follow the instructions below specific to the target platform.

Note: If the location of the GeoServer data directory is not set explicitly, the directory `data_dir` under the root of the GeoServer installation will be chosen by default.

10.2.1 Windows

On Windows platforms the location of the GeoServer data directory is controlled by the `GEOSERVER_DATA_DIR` environment variable.

To set the environment variable:

1. Open the System Control Panel.
2. Click *Advanced System Properties*.
3. Click *Environment Variables*.
4. Click the `New` button and create a environment variable called `GEOSERVER_DATA_DIR` and set it to the desired location.

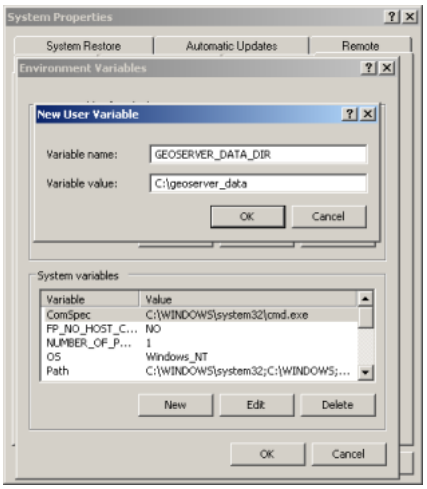


Fig. 10.1: Setting an environment variable on Windows

10.2.2 Linux

On Linux platforms the location of the GeoServer data directory is controlled by the `GEOSERVER_DATA_DIR` environment variable. Setting the variable can be achieved with the following command (in the terminal):

```
export GEOSERVER_DATA_DIR=/var/lib/geoserver_data
```

To make the variable persist, place the command in the `.bash_profile` or `.bashrc` file. Ensure that this done for the user running GeoServer.

10.2.3 Mac OS X

For the binary install of GeoServer on Mac OS X, the data directory is set in the same way as for Linux.

For the Mac OS X install, set the `GEOSERVER_DATA_DIR` environment variable to the desired directory location. See this page for details on how to set an environment variable in Mac OS X.

10.2.4 Web archive

When running a GeoServer WAR inside a servlet container, the data directory can be specified in a number of ways. The recommended method is to set a **servlet context parameter**. An alternative is to set a **Java system property**.

Context parameter

To specify the data directory using a servlet context parameter, create the following `<context-param>` element in the `WEB-INF/web.xml` file for the GeoServer application:

```
<web-app>
  ...
  <context-param>
    <param-name>GEOSERVER_DATA_DIR</param-name>
    <param-value>/var/lib/geoserver_data</param-value>
  </context-param>
  ...
</web-app>
```

Java system property

It is also possible to specify the data directory location with a Java system property. This method can be useful during upgrades, as it avoids the need to set the data directory after every upgrade.

Warning: Using a Java system property will typically set the property for all applications running in the servlet container, not just GeoServer.

The method of setting the Java system property is dependent on the servlet container:

- For **Tomcat**, edit the file `bin/setclasspath.sh` under the root of the Tomcat installation. Specify the `GEOSERVER_DATA_DIR` system property by setting the `CATALINA_OPTS` variable:

```
CATALINA_OPTS="-DGEOSERVER_DATA_DIR=/var/lib/geoserver_data"
```

- For **Glassfish**, edit the file `domains/<<domain>>/config/domain.xml` under the root of the Glassfish installation, where `<<domain>>` refers to the domain that the GeoServer web application is deployed under. Add a `<jvm-options>` element inside the `<java-config>` element:

```
...
<java-config>
   ...
  <jvm-options>-DGEOSERVER_DATA_DIR=/var/lib/geoserver_data</jvm-options>
</java-config>
...
```

10.2.5 Require files to exist

If the data directory is on a network filesystem, it can be desirable for security reasons to require one or more files or directories to exist before GeoServer will start, to prevent GeoServer from falling back into a default insecure configuration if the data directory appears to be empty because of the loss of this network resource.

To require files or directories to exist, use any of the methods above to set `GEOSERVER_REQUIRE_FILE`. Do not specify a mount point as this will still exist if a network filesystem is unavailable; instead specify a file or directory *inside* a network mount. For example:

Environment variable:

```
export GEOSERVER_REQUIRE_FILE=/mnt/server/geoserver_data/global.xml
```

Servlet context parameter:

```
<web-app>
  ...
  <context-param>
    <param-name>GEOSERVER_REQUIRE_FILE</param-name>
    <param-value>/mnt/server/geoserver_data/global.xml</param-value>
  </context-param>
  ...
</web-app>
```

Java system property:

```
CATALINA_OPTS="-DGEOSERVER_REQUIRE_FILE=/mnt/server/geoserver_data/global.xml"
```

Multiple files

To specify multiple files or directories that must exist, separate them with the path separator (`:` on Linux, `;` on Windows):

Environment variable:

```
export GEOSERVER_REQUIRE_FILE=/mnt/server/geoserver_data/global.xml:/mnt/server/data
```

Servlet context parameter:

```
<web-app>
  ...
  <context-param>
    <param-name>GEOSERVER_REQUIRE_FILE</param-name>
    <param-value>/mnt/server/geoserver_data/global.xml:/mnt/server/data</param-value>
  </context-param>
  ...
</web-app>
```

Java system property:

```
CATALINA_OPTS="-DGEOSERVER_REQUIRE_FILE=/mnt/server/geoserver_data/global.xml:/mnt/
↪server/data"
```

10.3 Structure of the data directory

This section gives an overview of the structure and contents of the GeoServer data directory.

This is not intended to be a complete reference to the GeoServer configuration information, since generally the data directory configuration files should not be accessed directly.

Instead, the *Web administration interface* can be used to view and modify the configuration, and for programmatic access and manipulation the *REST API* should be used.

The directories that do contain user-modifiable content are:

- `logs`
- `palettes`
- `templates`
- `user_projections`
- `www`

10.3.1 Top-level XML files

The top-level XML files contain information about the services and various global options for the server instance.

File	Description
`global.xml`	Contains settings common to all services, such as contact information, JAI settings, character sets and verbosity.
`logging.xml`	Specifies logging parameters, such as logging level, logfile location, and whether to log to stdout.
`wcs.xml`	Contains the service metadata and various settings for the WCS service.
`wfs.xml`	Contains the service metadata and various settings for the WFS service.
`wms.xml`	Contains the service metadata and various settings for the WMS service.

10.3.2 `workspaces`

The `workspaces` directory contain metadata about the layers published by GeoServer. It contains a directory for each defined **workspace**.

Each workspace directory contains directories for the **datastores** defined in it. Each datastore directory contains directories for the **layers** defined for the datastore.

Each layer directory contains a `layer.xml` file, and either a `coverage.xml` or a `featuretype.xml` file depending on whether the layer represents a raster or vector dataset.

10.3.3 data

The `data` directory can be used to store file-based geospatial datasets being served as layers.

Note: This should not be confused with the main GeoServer data directory, which is the parent to this directory.

This directory is commonly used to store shapefiles and raster files, but can be used for any data that is file-based.

The main benefit of storing data files under the `data` directory is portability.

Consider a shapefile stored external to the data directory at a location `C:\gis_data\foo.shp`. The `datastore` entry in `catalog.xml` for this shapefile would look like the following:

```
<datastore id="foo_shapefile">
   <connectionParams>
     <parameter name="url" value="file://C:/gis_data/foo.shp" />
   </connectionParams>
 </datastore>
```

Now consider trying to port this data directory to another host running GeoServer. The location `C:\gis_data\foo.shp` probably does not exist on the second host. So either the file must be copied to this location on the new host, or `catalog.xml` must be changed to reflect a new location.

This problem can be avoided by storing `foo.shp` in the `data` directory. In this case the `datastore` entry in `catalog.xml` becomes:

```
<datastore id="foo_shapefile">
  <connectionParams>
    <parameter name="url" value="file:data/foo.shp"/>
  </connectionParams>
</datastore>
```

The `value` attribute is rewritten to be relative to the `data` directory. This location independence allows the entire data directory to be copied to a new host and used directly with no additional changes.

10.3.4 demo

The `demo` directory contains files which define the sample requests available in the *Demo Request* page.

10.3.5 gwc

The `gwc` directory holds the tile cache created by the embedded *GeoWebCache* service.

10.3.6 layergroups

The `layergroups` directory contains configuration information for the defined layergroups.

10.3.7 logs

The `logs` directory contains configuration information for the various defined logging profiles, and the default `geoserver.log` log file.

Note: See also the *Logging* section for more details.

10.3.8 palettes

The `palettes` directory is used to store pre-computed **Image Palettes**. Image palettes are used by the GeoServer WMS as way to reduce the size of produced images while maintaining image quality.

Note: See also the *Paletted Images* tutorial for more information.

10.3.9 security

The `security` directory contains the files used to configure the GeoServer security subsystem. This includes a set of property files which define access roles, along with the services and data each role is authorized to access.

Note: See also the *Security* section for more information.

10.3.10 styles

The `styles` directory contains files which contain styling information used by the GeoServer WMS.

Note: See also the *Styling* section for more information.

For each SLD file in this directory there is a corresponding XML file:

```
<style>
  <id>StyleInfoImpl--570ae188:124761b8d78:-7fe1</id>
  <name>grass</name>
  <sldVersion>
    <version>1.0.0</version>
  </sldVersion>
  <filename>grass_poly.sld</filename>
  <legend>
    <width>32</width>
    <height>32</height>
    <format>image/png</format>
    <onlineResource>grass_fill.png</onlineResource>
```

```
  </legend>
</style>
```

The `styles` directory can also be used to host support files referenced during style configuration:

- Support files: SLD files can reference external graphics. This is useful when supplying your own icons in the form of image files or TrueType font files. Without any path information supplied, the default will be this directory.
- A style external graphic is dynamically created for use as a legend. The contents of the directory is published allowing clients to access the legends used. When running GeoServer on localhost, an image file `image.png` stored in this directory can be referenced in a browser using `http://<host:port>/geoserver/styles/image.png`.

10.3.11 templates

The `templates` directory contains files used by the GeoServer templating subsystem. Templates are used to customize the output of various GeoServer operations.

Note: See also *Freemarker Templates* for more information..

10.3.12 user_projections

The `user_projections` directory contains a file called `epsg.properties` which is used to define custom spatial reference systems that are not part of the official EPSG database.

Note: See also *Custom CRS Definitions* for more information.

10.3.13 www

The `www` directory is used to allow GeoServer to serve files like a regular web server. While not a replacement for a full web server, this can be useful for serving client-side mapping applications. The contents of this directory are served at `http://<host:port>/geoserver/www`.

Note: See also *Serving Static Files* for more information.

10.4 Migrating a data directory between versions

It is possible to keep the same data directory while migrating to different versions of GeoServer, such as during an update.

There should generally be no problems or issues migrating data directories between patch versions of GeoServer (for example, from 2.9.0 to 2.9.1 or vice versa).

Similarly, there should rarely be any issues involved with migrating between minor versions (for example, from 2.8.x to 2.9.x). **Care should be taken to back up the data directory prior to migration.**

Note: Some minor version migrations may not be reversible, since **newer versions of GeoServer may make backwards-incompatible changes** to the data directory.

10.5 Parameterize catalog settings

This feature allows to parameterize some of the settings in GeoServer's catalog.

What we mean with that is that you are able to use a templating mechanism to tailor GeoServer's settings to the environment in which is run.

For example, you'd want to move the latest changes you've made from a GeoServer instance running on machine **A** to another instance running on machine **B** but there are some differences in the way the two environments are set up, let's say the password used to connect to the database is different.

If you simply create a backup of the catalog from instance **A** and restore it on instance **B**, the stores configured on the database will not be accessible and the corresponding layers will not work properly.

Another example would be the max number of connections available in the connection pool to the database. You might want to have a different poll configuration in the two environments (maybe GeoServer on machine **A** is a test instance an GeoServer on machine **B** is used in production).

To overcome such limitation (to have the environment set up in the exact same way on the source machine and on the destination machine) the ENV parametrization allows you to customize the catalog configuration via the use of a templating system.

First of all to be able to use this feature, set the following flag via system variable to GeoServer's environment:

```
-DALLOW_ENV_PARAMETRIZATION=true
```

Then create a file called `geoserver-environment.properties` in the root of GeoServer's datadir. This file will contain the definitions for the variables parameterized in the catalog configuration.

Now edit GeoServer's configuration files of the source machine that you want to be parametric, for example let's parameterize the URL of a store (this can also be done via GeoServer admin UI):

```
vim coveragestore.xml
```

```
...
 <enabled>true</enabled>
  <workspace>
    <id>WorkspaceInfoImpl--134aa31e:1564c12ef68:-7ffe</id>
  </workspace>
  <__default>false</__default>
  <url>${store_url}</url>
</coverageStore>
```

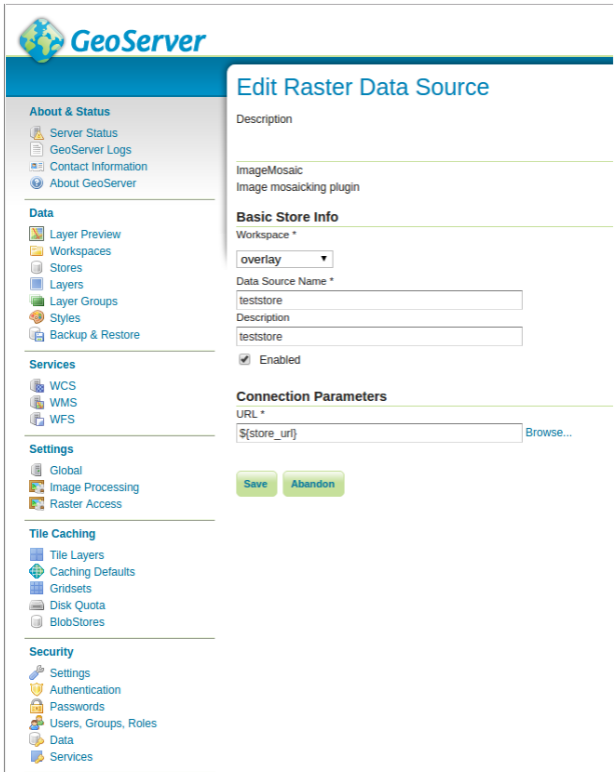
Add a definition for the variable **store_url** in your

```
geoserver-environment.properties
```

```
store_url = file:///var/geoserver/store/teststore
```

Now restart GeoServer and navigate to the store config

As you can see the URL in "Connection Parameters" settings now refers the variable **store_url** whose value is defined in the `geoserver-environment.properties` file.


GeoServer
About & Status
Server Status
GeoServer Logs
Contact Information
About GeoServer
Data
Layer Preview
Workspaces
Stores
Layers
Layer Groups
Styles
Backup & Restore
Services
WCS
WMS
WFS
Settings
Global
Image Processing
Raster Access
Tile Caching
Tile Layers
Caching Defaults
Gridsets
Disk Quota
BlobStores
Security
Settings
Authentication
Passwords
Users, Groups, Roles
Data
Services
Edit Raster Data Source
Description
ImageMosaic
Image mosaicking plugin
Basic Store Info
Workspace *
overlay
Data Source Name *
teststore
Description
teststore
Enabled
Connection Parameters
URL *
${store_url}
Browse...
Save
Abandon

CHAPTER 11

Running in a production environment

GeoServer is geared towards many different uses, from a simple test server to the enterprise-level data server. While many optimizations for GeoServer are set by default, here are some extra considerations to keep in mind when running GeoServer in a production environment.

11.1 Java Considerations

11.1.1 Use supported JRE

GeoServer's speed depends a lot on the chosen Java Runtime Environment (JRE). The latest versions of GeoServer are tested with both Oracle JRE and OpenJDK. Implementations other than those tested may work correctly, but are generally not recommended.

Tested:

- Java 11 - GeoServer 2.15.x and above (OpenJDK tested)
- Java 8 - GeoServer 2.9.x and above (OpenJDK and Oracle JRE tested)
- Java 7 - GeoServer 2.6.x to GeoServer 2.8.x (OpenJDK and Oracle JRE tested)
- Java 6 - GeoServer 2.3.x to GeoServer 2.5.x (Oracle JRE tested)
- Java 5 - GeoServer 2.2.x and earlier (Sun JRE tested)

For best performance we recommend the use *Oracle JRE 8* (also known as JRE 1.8).

As of GeoServer 2.0, a Java Runtime Environment (JRE) is sufficient to run GeoServer. GeoServer no longer requires a Java Development Kit (JDK).

11.1.2 Running on Java 11

GeoServer 2.15 will run under Java 11 with no additional configuration on **Tomcat 9** or newer and **Jetty 9.4.12** or newer.

Running GeoServer under Java 11 on other Application Servers may require some additional configuration. Some Application Servers do not support Java 11 yet.

- **Wildfly 14** supports Java 11, with some additional configuration - in the run configuration, under VM arguments add:

  –add-modules=java.se

  Future WildFly releases should support Java 11 with no additional configuration.
- **GlassFish** does not currently Java 11, although the upcoming 5.0.1 release is expected to include support for it.
- **WebLogic** do not yet support Java 11.

GeoServer cleanup

Once the installation is complete, you may optionally remove the original JAI files from the GeoServer `WEB-INF/lib` folder:

```
jai_core-x.y.z.jar
jai_imageio-x.y.jar
jai_codec-x.y.z.jar
```

where `x`, `y`, and `z` refer to specific version numbers.

11.1.3 Installing Unlimited Strength Jurisdiction Policy Files

These policy files are needed for unlimited cryptography. As an example, Java does not support AES with a key length of 256 bit. Installing the policy files removes these restrictions.

Open JDK

Since Open JDK is Open Source, the policy files are already installed.

Oracle Java

The policy files are available at

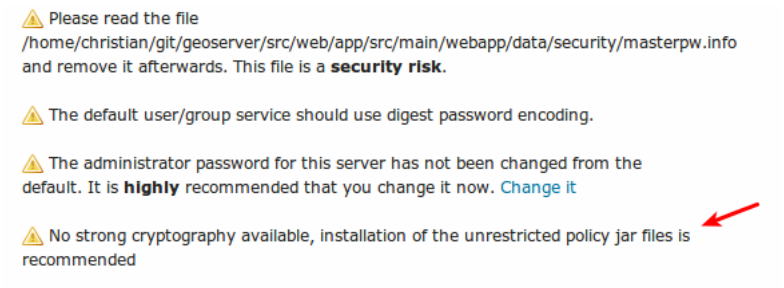
- Java 8 JCE policy jars
- Java 7 JCE policy jars
- Java 6 JCE policy jars

The download contains two files, **local_policy.jar** and **US_export_policy.jar**. The default versions of these two files are stored in JRE_HOME/lib/security. Replace these two files with the versions from the download.

Test if unlimited key length is available

Start or restart GeoServer and login as administrator. The annotated warning should have disappeared.

Additionally, the GeoServer log file should contain the following line:




```
"Strong cryptography is available"
```

Note: The replacement has to be done for each update of the Java runtime.

IBM Java

The policy files are available at

- IBM JCE policy jars

An IBM ID is needed to log in. The installation is identical to Oracle.

11.1.4 Outdated: install native JAI and ImageIO extensions

The Java Advanced Imaging API (JAI) is an advanced image processing library built by Oracle. GeoServer uses JAI-EXT, a set of replacement operations with bug fixes and NODATA support, for all image processing.

In case there is no interest in NODATA support, one can disable JAI-EXT and install the native JAI extensions to improve raster processing performance.

Warning: Users should take care that *JAI* native libraries remove support for NODATA pixels, provided instead by the pure Java JAI-EXT libraries.

Before installing native JAI, JAI-EXT can be disabled adding the following system variable to the JVM running GeoServer:

```
-Dorg.geotools.coverage.jaiext.enabled=false
```

Native JAI and ImageIO extensions are available for:

System	32-bit	64-bit
Windows	available	
Linux	available	available
Solaris	available	available
Max OSX		

Warning: A system installations of JAI and ImageIO may conflict with the pure java copy of JAI and ImageIO included in your GeoServer `WEB-INF/lib` folder - producing "class cast exceptions" preventing your application server from starting GeoServer.

- When installed as a "java extension" JAI and JAI ImageIO are unpacked into your JRE as both native code (in `bin`) and jars (in `ext/libs`). If you encounter this problem after installation of native the JAI and ImageIO extensions remove the pure java implementation from your GeoServer instances `WEB-INF/lib` folder:

```
rm jai_core-*jar jai_imageio-*.jar jai_codec-*.jar
```

- On OSX jars may be installed in `~/Library/Java/Extensions`, we advise removing these jars if present as they are no longer maintained by Apple.

Note: Native ImageIO encoding may not always be the best choice, we recommend the built-in *PNGJ based encoder* and *libjpeg-turbo Map Encoder Extension* for png8 and jpeg encoding performance.

Installing native JAI on Windows

1. Go to the JAI download page and download the Windows installer for version 1.1.3. At the time of writing only the 32 bit version of the installer is available, so if you are using a JDK, you will want to download jai-1_1_3-lib-windows-i586-jdk.exe, and if you are using a JRE, you will want to download jai-1_1_3-lib-windows-i586-jre.exe.
2. Run the installer and point it to the JDK/JRE install that GeoServer will use to run.
3. Go to the JAI Image I/O download page and download the Windows installer for version 1.1. At the time of writing only the 32 bit version of the installer is available, so if you are using a JDK, you will want to download jai_imageio-1_1-lib-windows-i586-jdk.exe, and if you are using a JRE, you will want to download jai_imageio-1_1-lib-windows-i586-jre.exe
4. Run the installer and point it to the JDK/JRE install that GeoServer will use to run.

Note: These installers are limited to allow adding native extensions to just one version of the JDK/JRE on your system. If native extensions are needed on multiple versions, manually unpacking the extensions will be necessary. See the section on *Installing native JAI manually*.

Note: These installers are also only able to apply the extensions to the currently used JDK/JRE. If native extensions are needed on a different JDK/JRE than that which is currently used, it will be necessary to uninstall the current one first, then run the setup program against the remaining JDK/JRE.

Installing native JAI on Linux

1. Go to the OpenGeo JAI download page and download the Linux installer for version 1.1.3, choosing the appropriate architecture:
   - *i586* for the 32 bit systems
   - *amd64* for the 64 bit ones (even if using Intel processors)

2. Copy the file into the directory containing the JDK/JRE and then run it. For example, on an Ubuntu 32 bit system:

```
$ sudo cp jai-1_1_3-lib-linux-i586-jdk.bin /usr/lib/jvm/java-6-sun
$ cd /usr/lib/jvm/java-6-sun
$ sudo sh jai-1_1_3-lib-linux-i586-jdk.bin
# accept license
$ sudo rm jai-1_1_3-lib-linux-i586-jdk.bin
```

3. Go to the OpenGeo JAI Image I/O Download page and download the Linux installer for version 1.1, choosing the appropriate architecture:
   - *i586* for the 32 bit systems
   - *amd64* for the 64 bit ones (even if using Intel processors)
4. Copy the file into the directory containing the JDK/JRE and then run it. If you encounter difficulties, you may need to export the environment variable `_POSIX2_VERSION=199209`. For example, on a Ubuntu 32 bit Linux system:

```
$ sudo cp jai_imageio-1_1-lib-linux-i586-jdk.bin /usr/lib/jvm/java-6-sun
$ cd /usr/lib/jvm/java-6-sun
$ sudo su
$ export _POSIX2_VERSION=199209
$ sh jai_imageio-1_1-lib-linux-i586-jdk.bin
# accept license
$ rm ./jai_imageio-1_1-lib-linux-i586-jdk.bin
$ exit
```

Installing native JAI manually

You can install the native JAI manually if you encounter problems using the above installers, or if you wish to install the native JAI for more than one JDK/JRE.

Please refer to the GeoTools page on JAI installation for details.

11.2 Container Considerations

Java web containers such as Tomcat or Jetty ship with configurations that allow for fast startup, but don't always deliver the best performance.

11.2.1 Optimize your JVM

Set the following performance settings in the Java virtual machine (JVM) for your container. These settings are not specific to any container.

Option	Description
`-Xms128m`	By starting with a larger heap GeoServer will not need to pause and ask the operating system for more memory during heavy load. The setting `-Xms128m` will tell the virtual machine to acquire grab a 128m heap memory on initial startup.
`-Xmx756M`	Defines an upper limit on how much heap memory Java will request from the operating system (use more if you have excess memory). By default, the JVM will use 1/4 of available system memory. The setting `-Xms756m` allocates 756MB of memory to GeoServer.
`-XX:SoftRefLRUPolicyMSPerMB=36000`	Increases the lifetime of "soft references" in GeoServer. GeoServer uses soft references to cache datastore, spatial reference systems, and other data structures. By increasing this value to `36000` (which is 36 seconds) these values will stay in memory longer increasing the effectiveness of the cache.
`-XX:+UseParallelGC`	The default garbage collector up to Java 8, **pauses the application while using several threads to recover memory**. Recommended if your GeoServer will be under light load and can tolerate pauses to clean up memory.
`-XX:+UseParNewGC`	Enables use of the concurrent mark sweep (CMS) garbage collector **uses multiple threads to recover memory while the application is running**. Recommended for GeoServer under continuous use, with heap sizes of less than 6GB.
`-XX:+UseG1GC`	The default garbage collector since Java 9. Enables use of the Garbage First Garbage Collector (G1) using **background threads to scan memory while the application is running** prior to cleanup. Recommended for GeoServer under continuous load and heap sizes of 6GB or more. Additionally you may experiment with `-XX:+UseStringDeduplicationJVM` to ask G1 to better manage common text strings in memory.

For more information about JVM configuration, see the article Performance tuning garbage collection in Java and The 4 Java Garbage Collectors.

Note: You can only use one garbage collector at a time. Please refrain from combining garbage collectors at runtime.

Note: If you're serving just vector data, you'll be streaming, so having more memory won't increase performance. If you're serving coverages, however, image processing will use a tile cache and benifit from more memory. As an administrator you can configure the portion of memory available as a tile cache (see the Server Config page in the *Web administration interface* section) - for example to use `0.75` to allocate `75%` of the heap as a tile cache.

Note: You can try out memory settings on the command line to check settings/defaults prior to use.

To check settings use `java -Xms128m -Xmx756m -XX:+PrintFlagsFinal -version | grep HeapSize`:

```
uintx InitialHeapSize    := 134217728      {product}
uintx MaxHeapSize        := 792723456      {product}
```

Which when converted from bytes matches `128` MB initial heap size, and `512` MB max heap size.

Check defaults for your hardware using `java -XX:+PrintFlagsFinal -version | grep HeapSize`:

```
uintx InitialHeapSize    := 268435456      {product}
uintx MaxHeapSize        := 4294967296     {product}
```

The above results (from a 16 GB laptop) amount to initial heap size of 256m, and a max heap size of around 4 GB (or around 1/4 of system memory).

11.2.2 Enable the Marlin rasterizer

The Marlin rasterizer in Java 8 and getting better performance and scalability while rendering vector data in Java 8. In order to enable it add the following among the JVM startup options:

```
-Xbootclasspath/a:$MARLIN_JAR
-Dsun.java2d.renderer=org.marlin.pisces.MarlinRenderingEngine
```

where `$MARLIN_JAR` is the location of the `marlin*.jar` file located in the geoserver/WEB-INF/lib directory.

11.2.3 Enable CORS

The standalone distributions of GeoServer include the Jetty application server. Enable Cross-Origin Resource Sharing (CORS) to allow JavaScript applications outside of your own domain to use GeoServer.

For more information on what this does and other options see Jetty Documentation

Uncomment the following <filter> and <filter-mapping> from `webapps/geoserver/WEB-INF/web.xml`:

```
<web-app>
  <filter>
      <filter-name>cross-origin</filter-name>
      <filter-class>org.eclipse.jetty.servlets.CrossOriginFilter</filter-class>
  </filter>
  <filter-mapping>
      <filter-name>cross-origin</filter-name>
      <url-pattern>/*</url-pattern>
  </filter-mapping>
 </web-app>
```

11.3 Configuration Considerations

11.3.1 Use production logging

Logging may visibly affect the performance of your server. High logging levels are often necessary to track down issues, but by default you should run with low levels. (You can switch the logging levels while

GeoServer is running.)

You can change the logging level in the *Web administration interface*. You'll want to choose the *PRODUCTION* logging configuration.

11.3.2 Set a service strategy

A service strategy is the method in which output is served to the client. This is a balance between proper form (being absolutely sure of reporting errors with the proper OGC codes, etc) and speed (serving output as quickly as possible). This is a decision to be made based on the function that GeoServer is providing. You can configure the service strategy by modifying the `web.xml` file of your GeoServer instance.

The possible strategies are:

Strategy	Description
`SPEED`	Serves output right away. This is the fastest strategy, but proper OGC errors are usually omitted.
`BUFFER`	Stores the whole result in memory, and then serves it after the output is complete. This ensures proper OGC error reporting, but delays the response quite a bit and can exhaust memory if the response is large.
`FILE`	Similar to `BUFFER`, but stores the whole result in a file instead of in memory. Slower than `BUFFER`, but ensures there won't be memory issues.
`PARTIAL-BUFFER`	A balance between `BUFFER` and `SPEED`, this strategy tries to buffer in memory a few KB of response, then serves the full output.

11.3.3 Personalize your server

This is isn't a performance consideration, but is just as important. In order to make GeoServer as useful as possible, you should customize the server's metadata to your organization. It may be tempting to skip some of the configuration steps, and leave in the same keywords and abstract as the sample, but this will only confuse potential users.

Suggestions:

- Fill out the WFS, WMS, and WCS Contents sections (this info will be broadcast as part of the capabilities documents)
- Serve your data with your own namespace (and provide a correct URI)
- Remove default layers (such as `topp:states`)

11.3.4 Configure service limits

Make sure clients cannot request an inordinate amount of resources from your server.

In particular:

- Set the maximum amount of features returned by each WFS GetFeature request (this can also be set on a per featuretype basis by modifying the `info.xml` files directly)
- Set the WMS `request limits` so that no request will consume too much memory or too much time

11.3.5 Set security

GeoServer includes support for WFS-T (transactions) by default, which lets users modify your data. If you don't want your database modified, you can turn off transactions in the the *Web administration interface*. Set the *Service Level* to `Basic`.

If you'd like some users to be able to modify some but not all of your data, you will have to set up an external security service. An easy way to accomplish this is to run two GeoServer instances and configure them differently, and use authentication to only allow certain users to have access.

For extra security, make sure that the connection to the datastore that is open to all is through a user who has read-only permissions. This will eliminate the possibility of a SQL injection (though GeoServer is generally not vulnerable to that sort of attack).

11.3.6 Cache your data

Server-side caching of WMS tiles is the best way to increase performance. In caching, pre-rendered tiles will be saved, eliminating the need for redundant WMS calls. There are several ways to set up WMS caching for GeoServer. GeoWebCache is the simplest method, as it comes bundled with GeoServer. (See the section on *GeoWebCache* for more details.) Another option is TileCache. You can also use a more generic caching system, such as OSCache (an embedded cache service) or Squid (a web cache proxy).

Caching is also possible for WFS layers, in a very limited fashion. For DataStores that don't have a quick way to determine feature counts (i.e. shapefiles), enabling caching can prevent querying a store twice during a single request. To enable caching, set the Java system property `org.geoserver.wfs.getfeature.cachelimit` to a positive integer. Any data sets that are smaller than the cache limit will be cached for the duration of a request, which will prevent them from being queried a second time for the feature count. Note that this may adversely affect some types of DataStores, as it bypasses any feature count optimizations that may exist.

11.3.7 Disable the GeoServer web administration interface

In some circumstances, you might want to completely disable the web administration interface. There are two ways of doing this:

- Set the Java system property GEOSERVER_CONSOLE_DISABLED to true by adding -DGEOSERVER_CONSOLE_DISABLED=true to your container's JVM options
- Remove all of the gs-web*-.jar files from WEB-INF/lib

11.3.8 X-Frame-Options Policy

In order to prevent clickjacking attacks GeoServer defaults to setting the X-Frame-Options HTTP header to SAMEORIGIN. This prevents GeoServer from being embedded into an iFrame, which prevents certain kinds of security vulnerabilities. See the OWASP Clickjacking entry for details.

If you wish to change this behavior you can do so through the following properties:

- geoserver.xframe.shouldSetPolicy: controls whether the X-Frame-Options filter should be set at all. Default is true.
- geoserver.xframe.policy: controls what the set the X-Frame-Options header to. Default is SAMEORIGIN valid options are DENY, SAMEORIGIN and ALLOW-FROM [uri]

These properties can be set either via Java system property, command line argument (-D), environment variable or web.xml init parameter.

11.4 Data Considerations

11.4.1 Use an external data directory

GeoServer comes with a built-in data directory. However, it is a good idea to separate the data from the application. Using an external data directory allows for much easier upgrades, since there is no risk of configuration information being overwritten. An external data directory also makes it easy to transfer your configuration elsewhere if desired. To point to an external data directory, you only need to edit the `web.xml` file. If you are new to GeoServer, you can copy (or just move) the data directory that comes with GeoServer to another location.

11.4.2 Use a spatial database

Shapefiles are a very common format for geospatial data. But if you are running GeoServer in a production environment, it is better to use a spatial database such as PostGIS. This is essential if doing transactions (WFS-T). Most spatial databases provide shapefile conversion tools. Although there are many options for spatial databases (see the section on *Databases*), PostGIS is recommended. Oracle, DB2, and ArcSDE are also supported.

11.4.3 Pick the best performing coverage formats

There are very significant differences between performance of the various coverage formats.

Serving big coverage data sets with good performance requires some knowledge and tuning, since usually data is set up for distribution and archival. The following tips try to provide you with a base knowledge of how data restructuring affects performance, and how to use the available tools to get optimal data serving performance.

Choose the right format

The first key element is choosing the right format. Some formats are designed for data exchange, others for data rendering and serving. A good data serving format is binary, allows for multi-resolution extraction, and provides support for quick subset extraction at native resolutions.

Examples of such formats are GeoTiff, ECW, JPEG 2000 and MrSid. ArcGrid on the other hand is an example of format that's particularly ill-suited for large dataset serving (it's text based, no multi-resolution, and we have to read it fully even to extract a data subset in the general case).

GeoServer supports MrSID, ECW and JPEG 2000 through the GDAL Image Format plugin. MrSID is the easiest to work with, as their reader is now available under a GeoServer compatible open source format. If you have ECW files you have several non-ideal options. If you are only using GeoServer for educational or non-profit purposes you can use the plugin for free. If not you need to buy a license, since it's server software. You could also use GDAL to convert it to MrSID or tiled GeoTiffs. If your files are JPEG 2000 you can use the utilities of ECW and MrSID software. But the fastest is Kakadu, which requires a license.

Setup Geotiff data for fast rendering

As soon as your Geotiffs gets beyond some tens of megabytes you'll want to add the following capabilities:

- inner tiling
- overviews

Inner tiling sets up the image layout so that it's organized in tiles instead of simple stripes (rows). This allows much quicker access to a certain area of the geotiff, and the GeoServer readers will leverage this by accessing only the tiles needed to render the current display area. The following sample command instructs gdal_translate to create a tiled geotiff.

```
gdal_translate -of GTiff -projwin -180 90 -50 -10 -co "TILED=YES" bigDataSet.ecw␣
↪myTiff.tiff
```

An overview is a downsampled version of the same image, that is, a zoomed out version, which is usually much smaller. When GeoServer needs to render the Geotiff, it'll look for the most appropriate overview as a starting point, thus reading and converting way less data. Overviews can be added using gdaladdo, or the the OverviewsEmbedded command included in Geotools. Here is a sample of using gdaladdo to add overviews that are downsampled 2, 4, 8 and 16 times compared to the original:

```
gdaladdo -r average mytiff.tif 2 4 8 16
```

As a final note, Geotiff supports various kinds of compression, but we do suggest to not use it. Whilst it allows for much smaller files, the decompression process is expensive and will be performed on each data access, significantly slowing down rendering. In our experience, the decompression time is higher than the pure disk data reading.

Handling huge data sets

If you have really huge data sets (several gigabytes), odds are that simply adding overviews and tiles does not cut it, making intermediate resolution serving slow. This is because tiling occurs only on the native resolution levels, and intermediate overviews are too big for quick extraction.

So, what you need is a way to have tiling on intermediate levels as well. This is supported by the ImagePyramid plugin.

This plugin assumes you have create various seamless image mosaics, each for a different resolution level of the original image. In the mosaic, tiles are actual files (for more info about mosaics, see the *ImageMosaic*). The whole pyramid structures looks like the following:

```
rootDirectory
    +- pyramid.properties
    +- 0
       +- mosaic metadata files
       +- mosaic_file_0.tiff
       +- ...
       +- mosiac_file_n.tiff
    +- ...
    +- 32
       +- mosaic metadata files
       +- mosaic_file_0.tiff
       +- ...
       +- mosiac_file_n.tiff
```

Creating a pyramid by hand can theoretically be done with gdal, but in practice it's a daunting task that would require some scripting, since gdal provides no "tiler" command to extract regular tiles out of an image, nor one to create a downsampled set of tiles. As an alternative, you can use the geotools PyramidBuilder tool (documentation on how to use this is pending, contact the developers if you need to use it).

11.5 Linux init scripts

You will have to adjust the scripts to your environment. Download a script, rename it to `geoserver` and move it to `/etc/init.d`. Use **`chmod`** to make the script executable and test with **`/etc/init.d/geoserver`**.

To set different values for environment variables, create a file `/etc/default/geoserver` and specify your environment.

Example settings in `/etc/default/geoserver` for your environment:

```
USER=geoserver
GEOSERVER_DATA_DIR=/home/$USER/data_dir
GEOSERVER_HOME=/home/$USER/geoserver
JAVA_HOME=/usr/lib/jvm/java-6-sun
JAVA_OPTS="-Xms128m -Xmx512m"
```

11.5.1 Debian/Ubuntu

`Download the init script`

11.5.2 Suse

`Download the init script`

11.5.3 Starting GeoServer in Tomcat

`Download the init script`

11.6 Other Considerations

11.6.1 Host your application separately

GeoServer includes a few sample applications in the demo section of the *Web administration interface*. For production instances, we recommend against this bundling of your application. To make upgrades and troubleshooting easier, please use a separate container for your application. It is perfectly fine, though, to use one container manager (such as Tomcat or Jetty) to host both GeoServer and your application.

11.6.2 Proxy your server

GeoServer can have the capabilities documents properly report a proxy. You can configure this in the Server configuration section of the *Web administration interface* and entering the URL of the external proxy in the field labeled `Proxy base URL`.

11.6.3 Publish your server's capabilities documents

In order to make it easier to find your data, put a link to your capabilities document somewhere on the web. This will ensure that a search engine will crawl and index it.

11.6.4 Set up clustering

Setting up a Cluster is one of the best ways to improve the reliability and performance of your GeoServer installation. All the most stable and high performance GeoServer instances are configured in some sort of cluster. There are a huge variety of techniques to configure a cluster, including at the container level, the virtual machine level, and the physical server level.

Andrea Aime is currently working on an overview of what some of the biggest GeoServer users have done, for his 'GeoServer in Production' talk at FOSS4G 2009. In time that information will be migrated to tutorials and white papers.

11.7 Troubleshooting

11.7.1 Checking WFS requests

It often happens that users report issues with hand made WFS requests not working as expected. In the majority of the cases the request is malformed, but GeoServer does not complain and just ignores the malformed part (this behaviour is the default to make older WFS clients work fine with GeoServer).

If you want GeoServer to validate most WFS XML request you can post it to the following URL:

```
http://host:port/geoserver/ows?strict=true
```

Any deviation from the required structure will be noted in an error message. The only request type that is not validated in any case is the INSERT one (this is a GeoServer own limitation).

11.7.2 Leveraging GeoServer own log

GeoServer can generate a quite extensive log of its operations in the `$GEOSERVER_DATA_DIR/logs/geoserver.log` file. Looking into such file is one of the first things to do when troubleshooting a problem, in particular it's interesting to see the log contents in correspondence of a misbehaving request. The amount of information logged can vary based on the logging profile chosen in the *Server Settings* configuration page.

11.7.3 Logging service requests

GeoServer provides a request logging filter that is normally inactive. The filter can log both the requested URL and POST requests contents. Normally it is disabled due to its overhead. If you need to have an history of the incoming requests you can enable it by changing the `geoserver/WEB-INF/web.xml` contents to look like:

```
<filter>
  <filter-name>Request Logging Filter</filter-name>
  <filter-class>org.geoserver.filters.LoggingFilter</filter-class>
  <init-param>
      <param-name>enabled</param-name>
      <param-value>true</param-value>
  </init-param>
  <init-param>
      <param-name>log-request-bodies</param-name>
      <param-value>true</param-value>
  </init-param>
</filter>
```

This will log both the requests and the bodies, resulting in something like the following:

```
08 gen 11:30:13 INFO [geoserver.filters] - 127.0.0.1 "GET /geoserver/wms?HEIGHT=330&
↪WIDTH=660&LAYERS=nurc%3AArc_Sample&STYLES=&SRS=EPSG%3A4326&FORMAT=image%2Fjpeg&
↪SERVICE=WMS&VERSION=1.1.1&REQUEST=GetMap&EXCEPTIONS=application%2Fvnd.ogc.se_
↪inimage&BBOX=-93.515625,-40.078125,138.515625,75.9375" "Mozilla/5.0 (X11; U; Linux␣
↪i686; it; rv:1.9.0.15) Gecko/2009102815 Ubuntu/9.04 (jaunty) Firefox/3.0.15" "http:/
↪/localhost:8080/geoserver/wms?service=WMS&version=1.1.0&request=GetMap&
↪layers=nurc:Arc_Sample&styles=&bbox=-180.0,-90.0,180.0,90.0&width=660&height=330&
↪srs=EPSG:4326&format=application/openlayers"
08 gen 11:30:13 INFO [geoserver.filters] - 127.0.0.1 "GET /geoserver/wms?HEIGHT=330&
↪WIDTH=660&LAYERS=nurc%3AArc_Sample&STYLES=&SRS=EPSG%3A4326&FORMAT=image%2Fjpeg&
↪SERVICE=WMS&VERSION=1.1.1&REQUEST=GetMap&EXCEPTIONS=application%2Fvnd.ogc.se_
↪inimage&BBOX=-93.515625,-40.078125,138.515625,75.9375" took 467ms
08 gen 11:30:14 INFO [geoserver.filters] - 127.0.0.1 "GET /geoserver/wms?
↪REQUEST=GetFeatureInfo&EXCEPTIONS=application%2Fvnd.ogc.se_xml&BBOX=-93.515625%2C-
↪40.078125%2C138.515625%2C75.9375&X=481&Y=222&INFO_FORMAT=text%2Fhtml&QUERY_
↪LAYERS=nurc%3AArc_Sample&FEATURE_COUNT=50&Layers=nurc%3AArc_Sample&Styles=&Srs=EPSG
↪%3A4326&WIDTH=660&HEIGHT=330&format=image%2Fjpeg" "Mozilla/5.0 (X11; U; Linux i686;␣
↪it; rv:1.9.0.15) Gecko/2009102815 Ubuntu/9.04 (jaunty) Firefox/3.0.15" "http://
↪localhost:8080/geoserver/wms?service=WMS&version=1.1.0&request=GetMap&
↪layers=nurc:Arc_Sample&styles=&bbox=-180.0,-90.0,180.0,90.0&width=660&height=330&
↪srs=EPSG:4326&format=application/openlayers"
08 gen 11:30:14 INFO [geoserver.filters] - 127.0.0.1 "GET /geoserver/wms?
↪REQUEST=GetFeatureInfo&EXCEPTIONS=application%2Fvnd.ogc.se_xml&BBOX=-93.515625%2C-
↪40.078125%2C138.515625%2C75.9375&X=481&Y=222&INFO_FORMAT=text%2Fhtml&QUERY_
↪LAYERS=nurc%3AArc_Sample&FEATURE_COUNT=50&Layers=nurc%3AArc_Sample&Styles=&Srs=EPSG
↪%3A4326&WIDTH=660&HEIGHT=330&format=image%2Fjpeg" took 314ms
```

11.7.4 Using JDK tools to get stack and memory dumps

The JDK contains three useful command line tools that can be used to gather information about GeoServer instances that are leaking memory or not performing as requested: `jps`, `jstack` and `jmap`.

All tools work against a live Java Virtual Machine, the one running GeoServer in particular. In order for them to work properly you'll have to run them with a user that has enough privileges to connect to the JVM process, in particular super user or the same user that's running the JVM usually have the required right.

jps

jps is a tool listing all the Java processing running. It can be used to retried the `pid` (process id) of the virtual machine that is running GeoServer. For example:

```
> jps -mlv

16235 org.apache.catalina.startup.Bootstrap start -Djava.util.logging.manager=org.
↪apache.juli.ClassLoaderLogManager -Djava.util.logging.config.file=/home/aaime/devel/
↪webcontainers/apache-tomcat-6.0.18/conf/logging.properties -Djava.endorsed.dirs=/
↪home/aaime/devel/webcontainers/apache-tomcat-6.0.18/endorsed -Dcatalina.base=/home/
↪aaime/devel/webcontainers/apache-tomcat-6.0.18 -Dcatalina.home=/home/aaime/devel/
↪webcontainers/apache-tomcat-6.0.18 -Djava.io.tmpdir=/home/aaime/devel/webcontainers/
↪apache-tomcat-6.0.18/temp
11521  -XX:MinHeapFreeRatio=10 -XX:MaxHeapFreeRatio=20 -Djava.library.path=/usr/lib/
↪jni -Dosgi.requiredJavaVersion=1.5 -XX:MaxPermSize=256m -Xms64m -Xmx1024m -
↪XX:CMSClassUnloadingEnabled -XX:CMSPermGenSweepingEnabled -XX:+UseParNewGC
16287 sun.tools.jps.Jps -mlv -Dapplication.home=/usr/lib/jvm/java-6-sun-1.6.0.16 -
↪Xms8m
```

The output shows the `pid`, the main class name if available, and the parameters passed to the JVM at startup. In this example `16235` is Tomcat hosting GeoServer, `11521` is an Eclipse instance, and `16287` is `jps` itself. In the common case you'll have only few JVM and the one running GeoServer can be identified by the parameters passed to it.

jstack

jstack is a tool extracting a the current stack trace for each thread running in the virtual machine. It can be used to identify scalability issues and to gather what the program is actually doing.

It usually takes people knowing about the inner workings of GeoServer can properly interpret the jstack output.

An example of usage:

```
> jstack -F -l 16235 > /tmp/tomcat-stack.txt
Attaching to process ID 16235, please wait...
Debugger attached successfully.
Server compiler detected.
JVM version is 14.2-b01
```

And the file contents might look like:

```
Deadlock Detection:

No deadlocks found.

Thread 16269: (state = BLOCKED)
 - java.lang.Object.wait(long) @bci=0 (Interpreted frame)
 - org.apache.tomcat.util.threads.ThreadPool$MonitorRunnable.run() @bci=10, line=565␣
↪(Interpreted frame)
 - java.lang.Thread.run() @bci=11, line=619 (Interpreted frame)

Locked ownable synchronizers:
    - None

Thread 16268: (state = IN_NATIVE)
 - java.net.PlainSocketImpl.socketAccept(java.net.SocketImpl) @bci=0 (Interpreted␣
↪frame)
 - java.net.PlainSocketImpl.accept(java.net.SocketImpl) @bci=7, line=390 (Interpreted␣
↪frame)
 - java.net.ServerSocket.implAccept(java.net.Socket) @bci=60, line=453 (Interpreted␣
↪frame)
 - java.net.ServerSocket.accept() @bci=48, line=421 (Interpreted frame)
 - org.apache.jk.common.ChannelSocket.accept(org.apache.jk.core.MsgContext) @bci=46,␣
↪line=306 (Interpreted frame)
 - org.apache.jk.common.ChannelSocket.acceptConnections() @bci=72, line=660␣
↪(Interpreted frame)
 - org.apache.jk.common.ChannelSocket$SocketAcceptor.runIt(java.lang.Object[]) @bci=4,
↪ line=870 (Interpreted frame)
 - org.apache.tomcat.util.threads.ThreadPool$ControlRunnable.run() @bci=167, line=690␣
↪(Interpreted frame)
 - java.lang.Thread.run() @bci=11, line=619 (Interpreted frame)

Locked ownable synchronizers:
    - None
```

```
Thread 16267: (state = BLOCKED)
 - java.lang.Object.wait(long) @bci=0 (Interpreted frame)
 - java.lang.Object.wait() @bci=2, line=485 (Interpreted frame)
 - org.apache.tomcat.util.threads.ThreadPool$ControlRunnable.run() @bci=26, line=662␣
↪(Interpreted frame)
 - java.lang.Thread.run() @bci=11, line=619 (Interpreted frame)

Locked ownable synchronizers:
    - None

...
```

jmap

jmap is a tool to gather information about the a Java virtual machine. It can be used in a few interesting ways.

By running it without arguments (past the pid of the JVM) it will print out a **dump of the native libraries used by the JVM**. This can come in handy when one wants to double check GeoServer is actually using a certain version of a native library (e.g., GDAL):

```
> jmap 17251

Attaching to process ID 17251, please wait...
Debugger attached successfully.
Server compiler detected.
JVM version is 14.2-b01
0x08048000      46K     /usr/lib/jvm/java-6-sun-1.6.0.16/jre/bin/java
0x7f87f000      6406K   /usr/lib/jvm/java-6-sun-1.6.0.16/jre/lib/i386/libNCSEcw.so.0
0x7f9b2000      928K    /usr/lib/libstdc++.so.6.0.10
0x7faa1000      7275K   /usr/lib/jvm/java-6-sun-1.6.0.16/jre/lib/i386/libgdal.so.1
0x800e9000      1208K   /usr/lib/jvm/java-6-sun-1.6.0.16/jre/lib/i386/libclib_jiio.so
0x80320000      712K    /usr/lib/jvm/java-6-sun-1.6.0.16/jre/lib/i386/libNCSUtil.so.0
0x80343000      500K    /usr/lib/jvm/java-6-sun-1.6.0.16/jre/lib/i386/libNCSCnet.so.0
0x8035a000      53K     /lib/libgcc_s.so.1
0x8036c000      36K     /usr/lib/jvm/java-6-sun-1.6.0.16/jre/lib/i386/libnio.so
0x803e2000      608K    /usr/lib/jvm/java-6-sun-1.6.0.16/jre/lib/i386/libawt.so
0x80801000      101K    /usr/lib/jvm/java-6-sun-1.6.0.16/jre/lib/i386/libgdaljni.so
0x80830000      26K     /usr/lib/jvm/java-6-sun-1.6.0.16/jre/lib/i386/headless/
↪libmawt.so
0x81229000      93K     /usr/lib/jvm/java-6-sun-1.6.0.16/jre/lib/i386/libnet.so
0xb7179000      74K     /usr/lib/jvm/java-6-sun-1.6.0.16/jre/lib/i386/libzip.so
0xb718a000      41K     /lib/tls/i686/cmov/libnss_files-2.9.so
0xb7196000      37K     /lib/tls/i686/cmov/libnss_nis-2.9.so
0xb71b3000      85K     /lib/tls/i686/cmov/libnsl-2.9.so
0xb71ce000      29K     /lib/tls/i686/cmov/libnss_compat-2.9.so
0xb71d7000      37K     /usr/lib/jvm/java-6-sun-1.6.0.16/jre/lib/i386/native_threads/
↪libhpi.so
0xb71de000      184K    /usr/lib/jvm/java-6-sun-1.6.0.16/jre/lib/i386/libjava.so
0xb7203000      29K     /lib/tls/i686/cmov/librt-2.9.so
0xb725d000      145K    /lib/tls/i686/cmov/libm-2.9.so
0xb7283000      8965K   /usr/lib/jvm/java-6-sun-1.6.0.16/jre/lib/i386/server/libjvm.so
0xb7dc1000      1408K   /lib/tls/i686/cmov/libc-2.9.so
0xb7f24000      9K      /lib/tls/i686/cmov/libdl-2.9.so
0xb7f28000      37K     /usr/lib/jvm/java-6-sun-1.6.0.16/jre/lib/i386/jli/libjli.so
0xb7f32000      113K    /lib/tls/i686/cmov/libpthread-2.9.so
```

```
0xb7f51000      55K     /usr/lib/jvm/java-6-sun-1.6.0.16/jre/lib/i386/libverify.so
0xb7f60000      114K    /lib/ld-2.9.so
```

It's also possible to get a **quick summary of the JVM heap status**:

```
> jmap -heap 17251

Attaching to process ID 17251, please wait...
Debugger attached successfully.
Server compiler detected.
JVM version is 14.2-b01

using thread-local object allocation.
Parallel GC with 2 thread(s)

Heap Configuration:
   MinHeapFreeRatio = 40
   MaxHeapFreeRatio = 70
   MaxHeapSize      = 778043392 (742.0MB)
   NewSize          = 1048576 (1.0MB)
   MaxNewSize       = 4294901760 (4095.9375MB)
   OldSize          = 4194304 (4.0MB)
   NewRatio         = 8
   SurvivorRatio    = 8
   PermSize         = 16777216 (16.0MB)
   MaxPermSize      = 67108864 (64.0MB)

Heap Usage:
PS Young Generation
Eden Space:
   capacity = 42401792 (40.4375MB)
   used     = 14401328 (13.734176635742188MB)
   free     = 28000464 (26.703323364257812MB)
   33.96396076845054% used
From Space:
   capacity = 4718592 (4.5MB)
   used     = 2340640 (2.232208251953125MB)
   free     = 2377952 (2.267791748046875MB)
   49.60462782118056% used
To Space:
   capacity = 4587520 (4.375MB)
   used     = 0 (0.0MB)
   free     = 4587520 (4.375MB)
   0.0% used
PS Old Generation
   capacity = 43188224 (41.1875MB)
   used     = 27294848 (26.0303955078125MB)
   free     = 15893376 (15.1571044921875MB)
   63.19974630121396% used
PS Perm Generation
   capacity = 38404096 (36.625MB)
   used     = 38378640 (36.60072326660156MB)
   free     = 25456 (0.0242767333984375MB)
   99.93371540369027% used
```

In the result it can be seen that the JVM is allowed to use up to 742MB of memory, and that at the moment the JVM is using 130MB (rough sum of the capacities of each heap section). In case of a persistent memory leak the JVM will end up using whatever is allowed to and each section of the heap will be almost 100%

used.

To see **how the memory is actually being used in a succinct way** the following command can be used (on Windows, replace `head -25` with `more`):

```
> jmap -histo:live 17251 | head -25

 num     #instances         #bytes  class name
----------------------------------------------
   1:         81668       10083280  <constMethodKlass>
   2:         81668        6539632  <methodKlass>
   3:         79795        5904728  [C
   4:        123511        5272448  <symbolKlass>
   5:          7974        4538688  <constantPoolKlass>
   6:         98726        3949040  org.hsqldb.DiskNode
   7:          7974        3612808  <instanceKlassKlass>
   8:          9676        2517160  [B
   9:          6235        2465488  <constantPoolCacheKlass>
  10:         10054        2303368  [I
  11:         83121        1994904  java.lang.String
  12:         27794        1754360  [Ljava.lang.Object;
  13:          9227         868000  [Ljava.util.HashMap$Entry;
  14:          8492         815232  java.lang.Class
  15:         10645         710208  [S
  16:         14420         576800  org.hsqldb.CachedRow
  17:          1927         574480  <methodDataKlass>
  18:          8937         571968  org.apache.xerces.dom.ElementNSImpl
  19:         12898         561776  [[I
  20:         23122         554928  java.util.HashMap$Entry
  21:         16910         541120  org.apache.xerces.dom.TextImpl
  22:          9898         395920  org.apache.xerces.dom.AttrNSImpl
```

By the dump we can see most of the memory is used by the GeoServer code itself (first 5 items) followed by the HSQL cache holding a few rows of the EPSG database. In case of a memory leak a few object types will hold the vast majority of the live heap. Mind, to look for a leak the dump should be gathered with the server almost idle. If, for example, the server is under a load of GetMap requests the main memory usage will be the byte[] holding the images while they are rendered, but that is not a leak, it's legitimate and temporary usage.

In case of memory leaks a developer will probably ask for a **full heap dump** to analyze with a high end profiling tool. Such dump can be generated with the following command:

```
> jmap -dump:live,file=/tmp/dump.hprof 17251
Dumping heap to /tmp/dump.hprof ...
Heap dump file created
```

The dump files are generally as big as the memory used so it's advisable to compress the resulting file before sending it to a developer.

11.7.5 XStream

GeoServer and GeoWebCache use XStream to read and write XML for configuration and for their REST APIs. In order to do this securely, it needs a list of Java classes that are safe to convert between objects and XML. If a class not on that list is given to XStream, it will generate the error `com.thoughtworks.xstream.security.ForbiddenClassException`. The specific class that was a problem should aslo be included. This may be a result of the lists of allowed classes missing a class, which should be reported

as a bug, or it may be caused by an extension/plugin not adding its classes to the list (finally, it could be someone trying to perform a "Remote Execution" attack, which is what the whitelist is designed to prevent).

This can be worked around by setting the system properties `GEOSERVER_XSTREAM_WHITELIST` for GeoServer or `GEOWEBCACHE_XSTREAM_WHITELIST` for GeoWebCache to a semicolon separated list of qualified class names. The class names may include wildcards `?` for a single character, `*` for any number of characters not including the separater `.`, and `**` for any number of characters including separators. For instance, `org.example.blah.SomeClass; com.demonstration.*; ca.test.**` will allow, the specific class `org.example.blah.SomeClass`, any class immediately within the package `com.demonstration`, and any class within the package `ca.test` or any of its descendant packages.

`GEOSERVER_XSTREAM_WHITELIST` and `GEOWEBCACHE_XSTREAM_WHITELIST` should only be used as a workarround until GeoServer, GWC, or the extension causing the problem has been updated, so please report to the users list the missing classes as soon as possible.

11.8 Make cluster nodes identifiable from the GUI

When running one or more clusters of GeoServer installations it is useful to identify which cluster (and eventually which node of the cluster) one is working against by just glancing the web administration UI.

This is possible by setting one variable, `GEOSERVER_NODE_OPTS`, with one of the supported mechanisms:

- as a system variable
- as an environment variable
- as a servlet context parameter

`GEOSERVER_NODE_OPTS` is a semicolumn separate list of key/value pairs and it can contain the following keys:

- id: the string identifying the node, which in turn can be a static string, or use the `$host_ip` and `$host_name` to report the host IP address or host name respectively
- color: the label color, as a CSS color
- background: the background color, as a CSS color

Here are some examples:

Fig. 11.1: *GEOSERVER_NODE_OPTS="id:test1;background:black;color:white"*

Fig. 11.2: *GEOSERVER_NODE_OPTS="id:$host_ip"*

Fig. 11.3: *GEOSERVER_NODE_OPTS="id:$host_name"*

CHAPTER 12

REST

GeoServer provides a RESTful interface through which clients can retrieve information about an instance and make configuration changes. Using the REST interface's simple HTTP calls, clients can configure GeoServer without needing to use the *Web administration interface*.

REST is an acronym for "REpresentational State Transfer". REST adopts a fixed set of operations on named resources, where the representation of each resource is the same for retrieving and setting information. In other words, you can retrieve (read) data in an XML format and also send data back to the server in similar XML format in order to set (write) changes to the system.

Operations on resources are implemented with the standard primitives of HTTP: GET to read; and PUT, POST, and DELETE to write changes. Each resource is represented as a URL, such as `http://GEOSERVER_HOME/rest/workspaces/topp`.

12.1 API

The following links provide direct access to the GeoServer REST API documentation, including definitions and examples of each endpoint:

- /about/manifests
- /datastores
- /coverages
- /coveragestores
- /featuretypes
- /fonts
- /layergroups
- /layers
- /monitoring

- /namespaces
- /services/wms | wfs | wcs/settings
- /reload
- /resource
- /security
- /settings
- /structuredcoverages
- /styles
- /templates
- /transforms
- /wmslayers
- /wmsstores
- /wmtslayers
- /wmtsstores
- /workspaces
- /usergroup
- /roles
- GeoWebCache:
  - /blobstores
  - /bounds
  - /diskquota
  - /filterupdate
  - /global
  - /gridsets
  - /index
  - /layers
  - /masstruncate
  - /statistics
  - /reload
  - /seed
- Importer extension:
  - /imports
  - /imports (tasks)
  - /imports (transforms)
  - /imports (data)
- Monitor extension:

  - /monitor
- XSLT extension:
  - /services/wfs/transforms

Note: You can also view the original *REST configuration API reference* section.

12.2 Examples

This section contains a number of examples which illustrate some of the most common uses of the REST API. They are grouped by endpoint.

12.2.1 About

The REST API allows you to set and retrieve information about the server itself.

Note: Read the *API reference for /about/manifests <manifests.yaml>__*.

Retrieving component versions

Retrieve the versions of the main components: GeoServer, GeoTools, and GeoWebCache

Request

curl

```
curl -v -u admin:geoserver -XGET -H "Accept: text/xml"
  http://localhost:8080/geoserver/rest/about/version.xml
```

Response

```
<about>
 <resource name="GeoServer">
  <Build-Timestamp>11-Dec-2012 17:55</Build-Timestamp>
  <Git-Revision>e66f8da85521f73d0fd00b71331069a5f49f7865</Git-Revision>
  <Version>2.3-SNAPSHOT</Version>
 </resource>
 <resource name="GeoTools">
  <Build-Timestamp>04-Dec-2012 02:31</Build-Timestamp>
  <Git-Revision>380a2b8545ee9221f1f2d38a4f10ef77a23bccae</Git-Revision>
  <Version>9-SNAPSHOT</Version>
 </resource>
 <resource name="GeoWebCache">
  <Git-Revision>2a534f91f6b99e5120a9eaa5db62df771dd01688</Git-Revision>
  <Version>1.3-SNAPSHOT</Version>
 </resource>
</about>
```

Retrieving manifests

Retrieve the full manifest and subsets of the manifest as known to the ClassLoader

Request

curl

```
curl -v -u admin:geoserver -XGET -H "Accept: text/xml"
  http://localhost:8080/geoserver/rest/about/manifest.xml
```

Note: The result will be a very long list of manifest information. While this can be useful, it is often desirable to filter this list.

Retrieve manifests, filtered by resource name

Note: This example will retrieve only resources where the `name` attribute matches `gwc-.*`.

Request

curl

```
curl -v -u admin:geoserver -XGET -H "Accept: text/xml"
  http://localhost:8080/geoserver/rest/about/manifest.xml?manifest=gwc-.*
```

Response

```
<about>
  <resource name="gwc-2.3.0">
    ...
  </resource>
  <resource name="gwc-core-1.4.0">
    ...
  </resource>
  <resource name="gwc-diskquota-core-1.4.0">
    ...
  </resource>
  <resource name="gwc-diskquota-jdbc-1.4.0">
    ...
  </resource>
  <resource name="gwc-georss-1.4.0">
    ...
  </resource>
  <resource name="gwc-gmaps-1.4.0">
    ...
  </resource>
  <resource name="gwc-kml-1.4.0">
    ...
  </resource>
  <resource name="gwc-rest-1.4.0">
    ...
  </resource>
```

```
  <resource name="gwc-tms-1.4.0">
    ...
  </resource>
  <resource name="gwc-ve-1.4.0">
    ...
  </resource>
  <resource name="gwc-wms-1.4.0">
    ...
  </resource>
  <resource name="gwc-wmts-1.4.0">
    ...
  </resource>
</about>
```

Retrieve manifests, filtered by resource property

Note: This example will retrieve only resources with a property equal to `GeoServerModule`.

Request

curl

```
curl -u admin:geoserver -XGET -H "Accept: text/xml"
  http://localhost:8080/geoserver/rest/about/manifest.xml?key=GeoServerModule
```

Response

```
<about>
 <resource name="control-flow-2.3.0">
  <GeoServerModule>extension</GeoServerModule>
  ...
 </resource>
 ...
 <resource name="wms-2.3.0">
  <GeoServerModule>core</GeoServerModule>
  ...
 </resource>
</about>
```

Retrieve manifests, filtered by both resource name and property

Note: This example will retrieve only resources where a property with named `GeoServerModule` has a value equal to `extension`.

Request

curl

```
curl -u admin:geoserver -XGET -H "Accept: text/xml"
  http://localhost:8080/geoserver/rest/about/manifest.xml?key=GeoServerModule&
↪value=extension
```

12.2.2 Fonts

The REST API allows you to list—but not modify—fonts available in GeoServer. It can be useful to use this operation to verify if a font used in a style file is available before uploading it.

Note: Read the API reference for /fonts.

Getting a list of all fonts

List all fonts on the server, in JSON format:

Request

curl

```
curl -u admin:geoserver -XGET http://localhost:8080/geoserver/rest/fonts.json
```

Response

```
{"fonts":["Calibri Light Italic","Microsoft PhagsPa Bold","Lucida Sans Typewriter␣
↪Oblique","ChaparralPro-Regular","Californian FB Italic"]}
```

List all fonts on the server, in XML format:

Request

curl

```
curl -u admin:geoserver -XGET http://localhost:8080/geoserver/rest/fonts.xml
```

Response

```
<root>
  <fonts>
    <entry>Calibri Light Italic</entry>
    <entry>Microsoft PhagsPa Bold</entry>
    <entry>Lucida Sans Typewriter Oblique</entry>
    <entry>ChaparralPro-Regular</entry>
    <entry>Californian FB Italic</entry>
  </fonts>
</root>
```

12.2.3 Layer groups

The REST API allows you to create and modify layer groups in GeoServer.

Note: The examples below specify global layer groups, but the examples will work in a workspace-specific construction as well.

Note: Read the API reference for /layergroups.

Creating a layer group

Create a new layer group based on already-published layers

Given the following content saved as `nycLayerGroup.xml`:

```
<layerGroup>
  <name>nyc</name>
  <layers>
    <layer>roads</layer>
    <layer>parks</layer>
    <layer>buildings</layer>
  </layers>
  <styles>
    <style>roads_style</style>
    <style>polygon</style>
    <style>polygon</style>
  </styles>
</layerGroup>
```

Request

curl

```
curl -v -u admin:geoserver -XPOST -d @nycLayerGroup.xml -H "Content-type: text/xml"
  http://localhost:8080/geoserver/rest/layergroups
```

Response

```
201 Created
```

Note: This layer group can be viewed with a WMS GetMap request:

```
http://localhost:8080/geoserver/wms/reflect?layers=nyc
```

12.2.4 Layers

The REST API allows you to list, create, upload, update, and delete layers in GeoServer.

Note: Read the API reference for /layers.

Listing all layers

List all layers on the server, in JSON format:

Request

curl

```
curl -u admin:geoserver -XGET http://localhost:8080/geoserver/rest/layers.json
```

Response

```
{
  "layers": {
     "layer": [
         {
             "name": "giant_polygon",
             "href": "http://localhost:8080/geoserver/rest/layers/giant_polygon.json"
         },
         {
             "name": "poi",
             "href": "http://localhost:8080/geoserver/rest/layers/poi.json"
         },
         ...
      ]
   }
}
```

List all styles in a workspace, in XML format:

Request

curl

```
curl -u admin:geoserver -XGET http://localhost:8080/geoserver/rest/layers.xml
```

Response

```
<layers>
  <layer>
    <name>giant_polygon</name>
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate" href="http://
↪localhost:8080/geoserver/rest/layers/giant_polygon.xml" type="application/xml"/>
  </layer>
  <layer>
    <name>poi</name>
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate" href="http://
↪localhost:8080/geoserver/rest/layers/poi.xml" type="application/xml"/>
  </layer>
  ...
</layers>
```

12.2.5 Security

The REST API allows you to adjust GeoServer security settings.

Note: Read the API reference for /security.

Listing the master password

Retrieve the master password for the "root" account

Request

curl

```
curl -v -u admin:geoserver -XGET http://localhost:8080/geoserver/rest/security/
↪masterpw.xml
```

Changing the master password

Change to a new master password

Note: Requires knowledge of the current master password.

Given a `changes.xml` file:

```
<masterPassword>
   <oldMasterPassword>-"}3a^Kh</oldMasterPassword>
   <newMasterPassword>geoserver1</newMasterPassword>
</masterPassword>
```

Request

curl

```
curl -v -u admin:geoserver -XPUT -H "Content-type: text/xml" -d @change.xml http://
↪localhost:8080/geoserver/rest/security/masterpw.xml
```

Response

```
200 OK
```

Listing the catalog mode

Fetch the current catalog mode

Request

curl

```
curl -v -u admin:geoserver -XGET   http://localhost:8080/geoserver/rest/security/acl/
↪catalog.xml
```

Response

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog>
    <mode>HIDE</mode>
</catalog>
```

Changing the catalog mode

Set a new catalog mode

Given a `newMode.xml` file:

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog>
    <mode>MIXED</mode>
</catalog>
```

Request

curl

```
curl -v -u admin:geoserver -XPUT -H "Content-type: text/xml" -d @newMode.xml http://
↪localhost:8080/geoserver/rest/security/acl/catalog.xml
```

Listing access control rules

Retrieve current list of access control rules

Request

curl

```
curl -v -u admin:geoserver -XGET http://localhost:8080/geoserver/rest/security/acl/
↪layers.xml
```

Response

```
<?xml version="1.0" encoding="UTF-8"?>
<rules />
```

Note: The above response shows no rules specified.

Changing access control rules

Set a new list of access control rules

Given a `rules.xml` file:

```
<?xml version="1.0" encoding="UTF-8"?>
<rules>
   <rule resource="topp.*.r">ROLE_AUTHORIZED</rule>
   <rule resource="topp.mylayer.w">ROLE_1,ROLE_2</rule>
</rules>
```

Request

curl

```
curl -v -u admin:geoserver -XPOST -H "Content-type: text/xml" -d @rules.xml http://
↪localhost:8080/geoserver/rest/security/acl/layers.xml
```

Response

```
201 Created
```

Deleting access control rules

Delete individual access control rule

Request

curl

```
curl -v -u admin:geoserver -XDELETE  http://localhost:8080/geoserver/rest/security/
↪acl/layers/topp.*.r
```

Response

```
200 OK
```

12.2.6 Styles

The REST API allows you to list, create, upload, update, and delete styles in GeoServer.

Note: Read the API reference for /styles.

Listing all styles

List all styles on the server, in JSON format:

Request

curl

```
curl -u admin:geoserver -XGET http://localhost:8080/geoserver/rest/styles.json
```

Response


```
{"styles":{"style":[{"name":"burg","href":"http:\/\/localhost:8080\/geoserver\/rest\/
↪styles\/burg.json"},{"name":"capitals","href":"http:\/\/localhost:8080\/geoserver\/
↪rest\/styles\/capitals.json"},{"name":"dem","href":"http:\/\/localhost:8080\/
↪geoserver\/rest\/styles\/dem.json"},{"name":"generic","href":"http:\/\/
↪localhost:8080\/geoserver\/rest\/styles\/generic.json"},{"name":"giant_polygon",
↪"href":"http:\/\/localhost:8080\/geoserver\/rest\/styles\/giant_polygon.json"},{
↪"name":"grass","href":"http:\/\/localhost:8080\/geoserver\/rest\/styles\/grass.json
↪"},{"name":"green","href":"http:\/\/localhost:8080\/geoserver\/rest\/styles\/green.
↪json"},{"name":"line","href":"http:\/\/localhost:8080\/geoserver\/rest\/styles\/
↪line.json"},{"name":"poi","href":"http:\/\/localhost:8080\/geoserver\/rest\/styles\/
↪poi.json"},{"name":"point","href":"http:\/\/localhost:8080\/geoserver\/rest\/
↪styles\/point.json"},{"name":"polygon","href":"http:\/\/localhost:8080\/geoserver\/
↪rest\/styles\/polygon.json"},{"name":"poly_landmarks","href":"http:\/\/
↪localhost:8080\/geoserver\/rest\/styles\/poly_landmarks.json"},{"name":"pophatch",
↪"href":"http:\/\/localhost:8080\/geoserver\/rest\/styles\/pophatch.json"},{"name":
↪"population","href":"http:\/\/localhost:8080\/geoserver\/rest\/styles\/population.
↪json"},{"name":"rain","href":"http:\/\/localhost:8080\/geoserver\/rest\/styles\/
↪rain.json"},{"name":"raster","href":"http:\/\/localhost:8080\/geoserver\/rest\/
↪styles\/raster.json"},{"name":"restricted","href":"http:\/\/localhost:8080\/
↪geoserver\/rest\/styles\/restricted.json"},{"name":"simple_roads","href":"http:\/\/
↪localhost:8080\/geoserver\/rest\/styles\/simple_roads.json"},{"name":"simple_streams
↪","href":"http:\/\/localhost:8080\/geoserver\/rest\/styles\/simple_streams.json"},{
↪"name":"tiger_roads","href":"http:\/\/localhost:8080\/geoserver\/rest\/styles\/
↪tiger_roads.json"}]}}
```


List all styles in a workspace, in XML format:

Request

curl

```
curl -u admin:geoserver -XGET http://localhost:8080/geoserver/rest/cite/styles.xml
```

Response


```
<styles>
  <style>
    <name>citerain</name>
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate" href="http://
↪localhost:8080/geoserver/rest/workspaces/cite/styles/citerain.xml" type=
↪"application/xml"/>
  </style>
</styles>
```


Retrieve a style

Download the actual style code for a style:

Request

curl

```
curl -u admin:geoserver -XGET http://localhost:8080/geoserver/rest/styles/rain.sld
```

Response

```
<StyledLayerDescriptor version="1.0.0" xmlns="http://www.opengis.net/sld" xmlns:ogc=
↪"http://www.opengis.net/ogc"
  xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xsi="http://www.w3.org/2001/
↪XMLSchema-instance"
  xsi:schemaLocation="http://www.opengis.net/sld http://schemas.opengis.net/sld/1.0.0/
↪StyledLayerDescriptor.xsd">
  <NamedLayer>
    <Name>rain</Name>
    <UserStyle>
      <Name>rain</Name>
      <Title>Rain distribution</Title>
      <FeatureTypeStyle>
        <Rule>
          <RasterSymbolizer>
            <Opacity>1.0</Opacity>
            <ColorMap>
              <ColorMapEntry color="#FF0000" quantity="0" />
              <ColorMapEntry color="#FFFFFF" quantity="100"/>
              <ColorMapEntry color="#00FF00" quantity="2000"/>
              <ColorMapEntry color="#0000FF" quantity="5000"/>
            </ColorMap>
          </RasterSymbolizer>
        </Rule>
      </FeatureTypeStyle>
    </UserStyle>
  </NamedLayer>
</StyledLayerDescriptor>
```

Creating a style

You can create a new style on the server in two ways. In the first way, the creation is done in two steps: the style entry is created in the catalog, and then the style content is uploaded. The second way can add the style to the server in a single step by uploading a ZIP containing the style content:

Create a new style in two steps:

Request

curl

```
curl -v -u admin:geoserver -XPOST -H "Content-type: text/xml" -d "<style><name>roads_
↪style</name><filename>roads.sld</filename></style>" http://localhost:8080/geoserver/
↪rest/styles
```

Response

```
201 Created
```

Request

curl

```
curl -v -u admin:geoserver -XPUT -H "Content-type: application/vnd.ogc.sld+xml" -d␣
↪@roads.sld http://localhost:8080/geoserver/rest/styles/roads_style
```

Response

```
200 OK
```

Create a new style in a single step:

Request

curl

```
curl -u admin:geoserver -XPOST -H "Content-type: application/zip" --data-binary␣
↪@roads_style.zip http://localhost:8080/geoserver/rest/styles
```

Response

```
201 Created
```

This example will create a new style on the server and populate it the contents of a local SLD file and related images provided in a SLD package. A SLD package is a zip file containing the SLD style and related image files used in the SLD.

The following creates a new style named `roads_style`.

Each code block below contains a single command that may be extended over multiple lines.

Request

curl

```
curl -u admin:geoserver -XPOST -H "Content-type: application/zip"
--data-binary @roads_style.zip
http://localhost:8080/geoserver/rest/styles
```

Response

```
201 OK
```

The SLD itself can be downloaded through a a GET request:

curl

```
curl -v -u admin:geoserver -XGET
http://localhost:8080/geoserver/rest/styles/roads_style.sld
```

Changing an existing style

Edit/reupload the content of an existing style on the server:

Request

curl

```
curl -u admin:geoserver -XPUT -H "Content-type: application/vnd.ogc.sld+xml" -d␣
↪@roads.sld
http://localhost:8080/geoserver/rest/styles/roads_style
```

Response

```
200 OK
```

Edit/reupload the content of an existing style on the server when the style is in a workspace:

Request

curl

```
curl -u admin:geoserver -XPUT -H "Content-type: application/vnd.ogc.sld+xml" -d␣
↪@roads.sld
http://localhost:8080/geoserver/rest/workspaces/cite/styles/roads_style
```

Response

```
200 OK
```

Edit/reupload the content of an existing style on the server using a ZIP file containing a shapefile:

Request

curl

```
curl -u admin:geoserver -XPUT -H "Content-type: application/zip" --data-binary @roads_
↪style.zip http://localhost:8080/geoserver/rest/styles/roads_style.zip
```

Response

```
200 OK
```

Deleting a style

Remove a style entry from the server, retaining the orphaned style content:

Request

curl

```
curl -u admin:geoserver -XDELETE http://localhost:8080/geoserver/rest/styles/zoning
```

Response

```
200 OK
```

Remove a style entry from the server, deleting the orphaned style content:

Request

curl

```
curl -u admin:geoserver -XDELETE http://localhost:8080/geoserver/rest/styles/zoning?
↪purge=true
```

Response

```
200 OK
```

12.2.7 Workspaces

The REST API allows you to create and manage workspaces in GeoServer.

Note: Read the API reference for /workspaces.

Adding a new workspace

Creates a new workspace named "acme" with a POST request

Request

curl

```
curl -v -u admin:geoserver -XPOST -H "Content-type: text/xml"
  -d "<workspace><name>acme</name></workspace>"
  http://localhost:8080/geoserver/rest/workspaces
```

python

TBD

java

TBD

Response

```
201 Created
```

Note: The `Location` response header specifies the location (URI) of the newly created workspace.

Listing workspace details

Retrieve information about a specific workspace

Request

curl

```
curl -v -u admin:geoserver -XGET -H "Accept: text/xml"
  http://localhost:8080/geoserver/rest/workspaces/acme
```

Note: The `Accept` header is optional.

python

TBD

java

TBD

Response

```
<workspace>
  <name>acme</name>
  <dataStores>
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate"
     href="http://localhost:8080/geoserver/rest/workspaces/acme/datastores.xml"
     type="application/xml"/>
  </dataStores>
  <coverageStores>
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate"
     href="http://localhost:8080/geoserver/rest/workspaces/acme/coveragestores.xml"
     type="application/xml"/>
  </coverageStores>
  <wmsStores>
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate"
     href="http://localhost:8080/geoserver/rest/workspaces/acme/wmsstores.xml"
     type="application/xml"/>
  </wmsStores>
</workspace>
```

This shows that the workspace can contain "`dataStores`" (for *vector data*), "`coverageStores`" (for *raster data*), and "`wmsStores`" (for *cascaded WMS servers*).

12.2.8 Stores

Uploading a shapefile

Create a new store "roads" by uploading a shapefile "roads.zip"

Request

curl

```
curl -v -u admin:geoserver -XPUT -H "Content-type: application/zip"
  --data-binary @roads.zip
  http://localhost:8080/geoserver/rest/workspaces/acme/datastores/roads/file.shp
```

python

TBD

java

TBD

Response

```
201 Created
```

Listing store details

*Retrieve information about a specific store**

Request

curl

```
curl -v -u admin:geoserver -XGET
  http://localhost:8080/geoserver/rest/workspaces/acme/datastores/roads.xml
```

python

TBD

java

TBD

Response

```
<dataStore>
  <name>roads</name>
  <type>Shapefile</type>
  <enabled>true</enabled>
  <workspace>
    <name>acme</name>
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate"
     href="http://localhost:8080/geoserver/rest/workspaces/acme.xml" type=
↪"application/xml"/>
  </workspace>
  <connectionParameters>
    <entry key="url">file:/C:/path/to/data_dir/data/acme/roads/</entry>
    <entry key="namespace">http://acme</entry>
  </connectionParameters>
  <__default>false</__default>
  <featureTypes>
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate"
     href="http://localhost:8080/geoserver/rest/workspaces/acme/datastores/roads/
↪featuretypes.xml"
     type="application/xml"/>
  </featureTypes>
</dataStore>
```

Request

Note: The XML response only provides details about the store itself, so you can use HTML to see the contents of the store.

```
curl -v -u admin:geoserver -XGET
  http://localhost:8080/geoserver/rest/workspaces/acme/datastores/roads.html
```

Listing featuretype details

Note: By default when a shapefile is uploaded, a featuretype is automatically created.

Request

curl

```
curl -v -u admin:geoserver -XGET
  http://localhost:8080/geoserver/rest/workspaces/acme/datastores/roads/featuretypes/
↪roads.xml
```

python

TBD

java

TBD

Response

```
<featureType>
  <name>roads</name>
  <nativeName>roads</nativeName>
  <namespace>
    <name>acme</name>
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate"
     href="http://localhost:8080/geoserver/rest/namespaces/acme.xml" type=
↪"application/xml"/>
  </namespace>
  ...
</featureType>
```

Adding an existing shapefile

Publish a shapefile "rivers.shp" that already exists on the server without needing to be uploaded

Request

curl

```
curl -v -u admin:geoserver -XPUT -H "Content-type: text/plain"
  -d "file:///data/shapefiles/rivers/rivers.shp"
  http://localhost:8080/geoserver/rest/workspaces/acme/datastores/rivers/external.shp
```

Note: The `external.shp` part of the request URI indicates that the file is coming from outside the catalog.

Response

```
201 Created
```

Adding a directory of existing shapefiles

Create a store containing a directory of shapefiles that already exists on the server without needing to be uploaded

Request

curl

```
curl -v -u admin:geoserver -XPUT -H "Content-type: text/plain"
  -d "file:///data/shapefiles/"
  "http://localhost:8080/geoserver/rest/workspaces/acme/datastores/shapefiles/
↪external.shp?configure=all"
```

Note: The `configure=all` query string parameter sets each shapefile in the directory to be loaded and published.

Response

```
201 Created
```

Adding a PostGIS database store

Add an existing PostGIS database named "nyc" as a new store

Note: This example assumes that a PostGIS database named `nyc` is present on the local system and is accessible by the user `bob`.

Given the following content saved as `nycDataStore.xml`:

```
<dataStore>
  <name>nyc</name>
  <connectionParameters>
    <host>localhost</host>
    <port>5432</port>
    <database>nyc</database>
    <user>bob</user>
    <passwd>postgres</passwd>
    <dbtype>postgis</dbtype>
  </connectionParameters>
</dataStore>
```

Request

curl

```
curl -v -u admin:geoserver -XPOST -T nycDataStore.xml -H "Content-type: text/xml"
   http://localhost:8080/geoserver/rest/workspaces/acme/datastores
```

Response

```
201 Created
```

Listing a PostGIS database store details

Retrieve information about a PostGIS store

Request

curl

```
curl -v -u admin:geoserver -XGET http://localhost:8080/geoserver/rest/workspaces/acme/
↪datastores/nyc.xml
```

Response

```
<dataStore>
  <name>nyc</name>
  <type>PostGIS</type>
  <enabled>true</enabled>
  <workspace>
    <name>acme</name>
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate"
     href="http://localhost:8080/geoserver/rest/workspaces/acme.xml" type=
↪"application/xml"/>
  </workspace>
  <connectionParameters>
    <entry key="port">5432</entry>
    <entry key="dbtype">postgis</entry>
    <entry key="host">localhost</entry>
    <entry key="user">bob</entry>
    <entry key="database">nyc</entry>
    <entry key="namespace">http://acme</entry>
  </connectionParameters>
  <__default>false</__default>
  <featureTypes>
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate"
     href="http://localhost:8080/geoserver/rest/workspaces/acme/datastores/nyc/
↪featuretypes.xml"
     type="application/xml"/>
  </featureTypes>
</dataStore>
```

Publishing a table from an existing PostGIS store

Publish a new featuretype from a PostGIS store table "buildings"

Note: This example assumes the table has already been created.

Request

curl

```
curl -v -u admin:geoserver -XPOST -H "Content-type: text/xml"
  -d "<featureType><name>buildings</name></featureType>"
  http://localhost:8080/geoserver/rest/workspaces/acme/datastores/nyc/featuretypes
```

Note: This layer can viewed with a WMS GetMap request:

```
http://localhost:8080/geoserver/wms/reflect?layers=acme:buildings
```

Creating a PostGIS table

Create a new featuretype in GeoServer and simultaneously create a table in PostGIS

Given the following content saved as `annotations.xml`:

```
<featureType>
  <name>annotations</name>
  <nativeName>annotations</nativeName>
  <title>Annotations</title>
  <srs>EPSG:4326</srs>
  <attributes>
    <attribute>
      <name>the_geom</name>
      <binding>org.locationtech.jts.geom.Point</binding>
    </attribute>
    <attribute>
      <name>description</name>
      <binding>java.lang.String</binding>
    </attribute>
    <attribute>
      <name>timestamp</name>
      <binding>java.util.Date</binding>
    </attribute>
  </attributes>
</featureType>
```

Request

curl

```
curl -v -u admin:geoserver -XPOST -T annotations.xml -H "Content-type: text/xml"
  http://localhost:8080/geoserver/rest/workspaces/acme/datastores/nyc/featuretypes
```

Note: The NYC store must be a PostGIS store for this to succeed.

Response

```
201 Created
```

A new and empty table named "annotations" in the "nyc" database will be created as well.

12.2.9 Uploading a new image mosaic

Upload a ZIP file containing a mosaic definition and granule(s)

Request

curl

```
curl -u admin:geoserver -XPUT -H "Content-type:application/zip" --data-binary␣
↪@polyphemus.zip
   http://localhost:8080/geoserver/rest/workspaces/topp/coveragestores/polyphemus/
↪file.imagemosaic
```

Response

```
200 OK
```

12.2.10 Updating an image mosaic contents

Harvest (or reharvest) a single file into the mosaic and update the mosaic index

Request

curl

```
curl -v -u admin:geoserver -XPOST -H "Content-type: text/plain" -d "file:///path/to/
↪the/file/polyphemus_20130302.nc"
   "http://localhost:8080/geoserver/rest/workspaces/topp/coveragestores/poly-
↪incremental/external.imagemosaic"
```

Response

```
201 Created
```

Harvest (or reharvest) a whole directory into the mosaic and update the mosaic index

Request

curl

```
curl -v -u admin:geoserver -XPOST -H "Content-type: text/plain" -d "file:///path/to/
↪the/mosaic/folder"
   "http://localhost:8080/geoserver/rest/workspaces/topp/coveragestores/poly-
↪incremental/external.imagemosaic"
```

Response

```
201 Created
```

12.2.11 Listing image mosaic details

Retrieve the image mosaic index structure

Request

curl

```
curl -v -u admin:geoserver -XGET "http://localhost:8080/geoserver/rest/workspaces/
↪topp/coveragestores/polyphemus-v1/coverages/NO2/index.xml"
```

Response

```
  <Schema>
 <attributes>
   <Attribute>
     <name>the_geom</name>
     <minOccurs>0</minOccurs>
     <maxOccurs>1</maxOccurs>
     <nillable>true</nillable>
     <binding>org.locationtech.jts.geom.Polygon</binding>
   </Attribute>
   <Attribute>
     <name>location</name>
     <minOccurs>0</minOccurs>
     <maxOccurs>1</maxOccurs>
     <nillable>true</nillable>
     <binding>java.lang.String</binding>
   </Attribute>
   <Attribute>
     <name>imageindex</name>
     <minOccurs>0</minOccurs>
     <maxOccurs>1</maxOccurs>
     <nillable>true</nillable>
     <binding>java.lang.Integer</binding>
   </Attribute>
   <Attribute>
     <name>time</name>
     <minOccurs>0</minOccurs>
     <maxOccurs>1</maxOccurs>
     <nillable>true</nillable>
     <binding>java.sql.Timestamp</binding>
   </Attribute>
   <Attribute>
     <name>elevation</name>
     <minOccurs>0</minOccurs>
     <maxOccurs>1</maxOccurs>
     <nillable>true</nillable>
     <binding>java.lang.Double</binding>
   </Attribute>
   <Attribute>
     <name>fileDate</name>
     <minOccurs>0</minOccurs>
     <maxOccurs>1</maxOccurs>
     <nillable>true</nillable>
     <binding>java.sql.Timestamp</binding>
   </Attribute>
   <Attribute>
     <name>updated</name>
     <minOccurs>0</minOccurs>
     <maxOccurs>1</maxOccurs>
     <nillable>true</nillable>
     <binding>java.sql.Timestamp</binding>
   </Attribute>
 </attributes>
 <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate" href="http://
↪localhost:8080/geoserver/rest/workspaces/topp/coveragestores/polyphemus-v1/
↪coverages/NO2/index/granules.xml" type="application/xml"/>
</Schema>
```

Retrieve the existing granule information

Request

curl

```
curl -v -u admin:geoserver -XGET "http://localhost:8080/geoserver/rest/workspaces/
↪topp/coveragestores/polyphemus-v1/coverages/NO2/index/granules.xml?limit=2"
```

Response

```
<?xml version="1.0" encoding="UTF-8"?>
<wfs:FeatureCollection xmlns:gf="http://www.geoserver.org/rest/granules" xmlns:ogc=
↪"http://www.opengis.net/ogc" xmlns:wfs="http://www.opengis.net/wfs" xmlns:gml=
↪"http://www.opengis.net/gml">
  <gml:boundedBy>
    <gml:Box srsName="http://www.opengis.net/gml/srs/epsg.xml#4326">
      <gml:coord>
        <gml:X>5.0</gml:X>
        <gml:Y>45.0</gml:Y>
      </gml:coord>
      <gml:coord>
        <gml:X>14.875</gml:X>
        <gml:Y>50.9375</gml:Y>
      </gml:coord>
    </gml:Box>
  </gml:boundedBy>
  <gml:featureMember>
    <gf:NO2 fid="NO2.1">
      <gf:the_geom>
        <gml:Polygon>
          <gml:outerBoundaryIs>
            <gml:LinearRing>
              <gml:coordinates>5.0,45.0 5.0,50.9375 14.875,50.9375 14.875,45.0 5.0,45.
↪0</gml:coordinates>
            </gml:LinearRing>
          </gml:outerBoundaryIs>
        </gml:Polygon>
      </gf:the_geom>
      <gf:location>polyphemus_20130301.nc</gf:location>
      <gf:imageindex>336</gf:imageindex>
      <gf:time>2013-03-01T00:00:00Z</gf:time>
      <gf:elevation>10.0</gf:elevation>
      <gf:fileDate>2013-03-01T00:00:00Z</gf:fileDate>
      <gf:updated>2013-04-11T10:54:31Z</gf:updated>
    </gf:NO2>
  </gml:featureMember>
  <gml:featureMember>
    <gf:NO2 fid="NO2.2">
      <gf:the_geom>
        <gml:Polygon>
          <gml:outerBoundaryIs>
            <gml:LinearRing>
              <gml:coordinates>5.0,45.0 5.0,50.9375 14.875,50.9375 14.875,45.0 5.0,45.
↪0</gml:coordinates>
            </gml:LinearRing>
          </gml:outerBoundaryIs>
        </gml:Polygon>
      </gf:the_geom>
```

```
      <gf:location>polyphemus_20130301.nc</gf:location>
      <gf:imageindex>337</gf:imageindex>
      <gf:time>2013-03-01T00:00:00Z</gf:time>
      <gf:elevation>35.0</gf:elevation>
      <gf:fileDate>2013-03-01T00:00:00Z</gf:fileDate>
      <gf:updated>2013-04-11T10:54:31Z</gf:updated>
    </gf:NO2>
  </gml:featureMember>
</wfs:FeatureCollection>
```

12.2.12 Removing image mosaic granules

Remove all the granules originating from a particular file

Request

curl

```
curl -v -u admin:geoserver -XDELETE "http://localhost:8080/geoserver/rest/workspaces/
↪topp/coveragestores/polyphemus-v1/coverages/NO2/index/granules.xml?filter=location=
↪'polyphemus_20130301.nc'"
```

Response

```
200 OK
```

12.2.13 Uploading an empty mosaic

Upload an archive with the definition of an mosaic, but with no granules

Given a `empty.zip` file containing:

- `datastore.properties` (PostGIS connection parameters)
- `indexer.xml` (Mosaic indexer; note the `CanBeEmpty=true` parameter)
- `polyphemus-test.xml` (Auxiliary file used by the NetCDF reader to parse schemas and tables)

Warning: Make sure to update the `datastore.properties` file with your connection parameters and refresh the ZIP before uploading it.

Request

curl

```
curl -u admin:geoserver -XPUT -H "Content-type:application/zip" --data-binary @empty.
↪zip
   http://localhost:8080/geoserver/rest/workspaces/topp/coveragestores/empty/file.
↪imagemosaic?configure=none
```

Note: The `configure=none` parameter allows for future configuration after harvesting.

Response

```
200 OK
```

Configure a coverage on the mosaic

Given a `coverageconfig.xml`:

```
<coverage>
  <nativeCoverageName>NO2</nativeCoverageName>
  <name>NO2</name>
</coverage>
```

Request

curl

```
curl -v -u admin:geoserver -XPOST -H "Content-type: text/xml" -d @"/path/to/
↪coverageconfig.xml" "http://localhost:8080/geoserver/rest/workspaces/topp/
↪coveragestores/empty/coverages"
```

Note: When specifying only the coverage name, the coverage will be automatically configured.

Response

```
201 Created
```

12.2.14 Uploading an app-schema mapping file

Create a new app-schema store and update the feature type mappings of an existing app-schema store by uploading a mapping configuration file

Note: The following request uploads an app-schema mapping file called `LandCoverVector.xml` to a data store called `LandCoverVector`. If no `LandCoverVector` data store existed in workspace `lcv` prior to the request, it would be created.

Request

curl

```
curl -v -X PUT -d @LandCoverVector.xml -H "Content-Type: text/xml"
-u admin:geoserver http://localhost:8080/geoserver/rest/workspaces/lcv/datastores/
↪LandCoverVector/file.appschema?configure=all
```

Response

```
201 Created
```

12.2.15 Listing app-schema store details

Request

curl

```
curl -v -u admin:geoserver -X GET
http://localhost:8080/geoserver/rest/workspaces/lcv/datastores/LandCoverVector.xml
```

Response

```
<dataStore>
  <name>LandCoverVector</name>
  <type>Application Schema DataAccess</type>
  <enabled>true</enabled>
  <workspace>
    <name>lcv</name>
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate" href="http://
↪localhost:8080/geoserver/rest/workspaces/lcv.xml" type="application/xml"/>
  </workspace>
  <connectionParameters>
    <entry key="dbtype">app-schema</entry>
    <entry key="namespace">http://inspire.ec.europa.eu/schemas/lcv/3.0</entry>
    <entry key="url">file:/path/to/data_dir/data/lcv/LandCoverVector/LandCoverVector.
↪appschema</entry>
  </connectionParameters>
  <__default>false</__default>
  <featureTypes>
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate" href="http://
↪localhost:8080/geoserver/rest/workspaces/lcv/datastores/LandCoverVector/
↪featuretypes.xml" type="application/xml"/>
  </featureTypes>
</dataStore>
```

12.2.16 Uploading a new app-schema mapping configuration file

Upload a new mapping configuration, stored in the mapping file "'LandCoverVector_alternative.xml", to the "LandCoverVector" data store

Request

curl

```
curl -v -X PUT -d @LandCoverVector_alternative.xml -H "Content-Type: text/xml"
  -u admin:geoserver http://localhost:8080/geoserver/rest/workspaces/lcv/datastores/
↪LandCoverVector/file.appschema?configure=none
```

Response

```
200 OK
```

Note: This time the `configure` parameter is set to `none`, because we don't want to configure again the feature types, just replace their mapping configuration.

Note: If the set of feature types mapped in the new configuration file differs from the set of feature types mapped in the old one (either some are missing, or some are new, or both), the best way to proceed is to delete the data store and create it anew issuing another PUT request, *as shown above*.

12.2.17 Uploading multiple app-schema mapping files

Create a new app-schema data store based on a complex mapping configuration split into multiple files, and show how to upload application schemas (i.e. XSD files) along with the mapping configuration.

Note: In the previous example, we have seen how to create a new app-schema data store by uploading a mapping configuration stored in a single file; this time, things are more complicated, since the mappings have been spread over two configuration files: the main configuration file is called `geosciml.appschema` and contains the mappings for three feature types: `GeologicUnit`, `MappedFeature` and `GeologicEvent`; the second file is called `cgi_termvalue.xml` and contains the mappings for a single non-feature type, `CGI_TermValue`.

Note: As explained in the *REST API reference documentation for data stores*, when the mapping configuration is spread over multiple files, the extension of the main configuration file must be `.appschema`.

The main configuration file includes the second file:

```
...
<includedTypes>
  <Include>cgi_termvalue.xml</Include>
</includedTypes>
...
```

We also want to upload to GeoServer the schemas required to define the mapping, instead of having GeoServer retrieve them from the internet (which is especially useful in case our server doesn't have access to the web). The main schema is called `geosciml.xsd` and is referred to in `geosciml.appschema` as such:

```
...
<targetTypes>
  <FeatureType>
    <schemaUri>geosciml.xsd</schemaUri>
  </FeatureType>
</targetTypes>
...
```

In this case, the main schema depends on several other schemas:

```
<include schemaLocation="geologicUnit.xsd"/>
<include schemaLocation="borehole.xsd"/>
<include schemaLocation="vocabulary.xsd"/>
<include schemaLocation="geologicRelation.xsd"/>
<include schemaLocation="fossil.xsd"/>
<include schemaLocation="value.xsd"/>
<include schemaLocation="geologicFeature.xsd"/>
<include schemaLocation="geologicAge.xsd"/>
<include schemaLocation="earthMaterial.xsd"/>
<include schemaLocation="collection.xsd"/>
<include schemaLocation="geologicStructure.xsd"/>
```

They don't need to be listed in the `targetTypes` section of the mapping configuration, but they must be included in the ZIP archive that will be uploaded.

Note: The GeoSciML schemas listed above, as pretty much any application schema out there, reference the base GML schemas (notably, `http://schemas.opengis.net/gml/3.1.1/base/gml.xsd`) and a few other remotely hosted schemas (e.g. `http://www.geosciml.org/cgiutilities/1.0/xsd/cgiUtilities.xsd`). For the example to work in a completely offline environment, one would have to either replace all remote references with local ones, or pre-populate the app-schema cache with a copy of the remote schemas. *GeoServer's user manual* contains more information on the app-schema cache.

To summarize, we'll upload to GeoServer a ZIP archive with the following contents:

```
geosciml.appschema      # main mapping file
cgi_termvalue.xml       # secondary mapping file
geosciml.xsd            # main schema
borehole.xsd
collection.xsd
earthMaterial.xsd
fossil.xsd
geologicAge.xsd
geologicFeature.xsd
geologicRelation.xsd
geologicStructure.xsd
geologicUnit.xsd
value.xsd
vocabulary.xsd
```

Request

curl

```
curl -X PUT --data-binary @geosciml.zip -H "Content-Type: application/zip"
-u admin:geoserver http://localhost:8080/geoserver/rest/workspaces/gsml/datastores/
↪geosciml/file.appschema?configure=all
```

Response

```
200 OK
```

A new `geosciml` data store will be created with three feature types in it:

```
<featureTypes>
  <featureType>
    <name>MappedFeature</name>
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate" href="http://
↪localhost:8080/geoserver/rest/workspaces/gsml/datastores/geosciml/featuretypes/
↪MappedFeature.xml" type="application/xml"/>
  </featureType>
  <featureType>
    <name>GeologicEvent</name>
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate" href="http://
↪localhost:8080/geoserver/rest/workspaces/gsml/datastores/geosciml/featuretypes/
↪GeologicEvent.xml" type="application/xml"/>
  </featureType>
  <featureType>
    <name>GeologicUnit</name>
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate" href="http://
↪localhost:8080/geoserver/rest/workspaces/gsml/datastores/geosciml/featuretypes/
↪GeologicUnit.xml" type="application/xml"/>
  </featureType>
</featureTypes>
```

12.2.18 REST configuration API reference

This section describes the GeoServer REST configuration API.

API details

This page contains information on the REST API architecture.

Authentication

REST requires that the client be authenticated. By default, the method of authentication used is Basic authentication. See the *Security* section for how to change the authentication method.

Status codes

An HTTP request uses a status code to relay the outcome of the request to the client. Different status codes are used for various purposes throughout this document. These codes are described in detail by the HTTP specification.

The most common status codes are listed below, along with their descriptions:

Status code	Description	Notes
200	OK	The request was successful
201	Created	A new resource was successfully created, such as a new feature type or data store
403	Forbidden	Often denotes a permissions mismatch
404	Not Found	Endpoint or resource was not at the indicated location
405	Method Not Allowed	Often denotes an endpoint accessed with an incorrect operation (for example, a GET request where a PUT/POST is indicated)
500	Internal Server Error	Often denotes a syntax error in the request

Formats and representations

A `format` specifies how a particular resource should be represented. A format is used:

- In an operation to specify what representation should be returned to the client
- In a POST or PUT operation to specify the representation being sent to the server

In a **GET** operation the format can be specified in two ways.

There are two ways to specify the format for a GET operation. The first option uses the `Accept` header. For example, with the header set to `"Accept: text/xml"` the resource would be returned as XML. The second option of setting the format is via a file extension. For example, given a resource `foo`, to request a representation of `foo` as XML, the request URI would end with `/foo.xml`. To request a representation as JSON, the request URI would end with `/foo.json`. When no format is specified the server will use its own internal format, usually HTML. When the response format is specified both by the header and by the extension, the format specified by the extension takes precedence.

In a **POST** or **PUT** operation, the format of content being sent to the server is specified with the `Content-type` header. For example, to send a representation in XML, use `"Content-type: text/xml"` or `"Content-type: application/xml"`. As with GET requests, the representation of the content returned from the server is specified by the `Accept` header or by the format.

The following table defines the `Content-type` values for each format:

Format	Content-type
XML	`application/xml`
JSON	`application/json`
HTML	`application/html`
SLD	`application/vnd.ogc.sld+xml`
ZIP	`application/zip`

Global settings

Allows access to GeoServer's global settings.

`/settings[.<format>]`

Controls all global settings.

Method	Action	Status code	Formats	Default Format
GET	List all global settings	200	HTML, XML, JSON	HTML
POST		405		
PUT	Update global settings	200	XML, JSON	
DELETE		405		

`/settings/contact[.<format>]`

Controls global contact information only.

Method	Action	Status code	Formats	Default Format
GET	List global contact information	200	HTML, XML, JSON	HTML
POST		405		
PUT	Update global contact	200	XML, JSON	
DELETE		405		

Workspaces

A `workspace` is a grouping of data stores. Similar to a namespace, it is used to group data that is related in some way.

`/workspaces[.<format>]`

Controls all workspaces.

Method	Action	Status code	Formats	Default Format
GET	List all workspaces	200	HTML, XML, JSON	HTML
POST	Create a new workspace	201 with `Location` header	XML, JSON	
PUT		405		
DELETE		405		

`/workspaces/<ws>[.<format>]`

Controls a specific workspace.

Method	Action	Status code	Formats	Default Format	Parameters
GET	Return workspace `ws`	200	HTML, XML, JSON	HTML	*quietOnNotFound*
POST		405			
PUT	200	Modify workspace `ws`	XML, JSON		
DELETE	200	Delete workspace `ws`	XML, JSON		*recurse*

Exceptions

Exception	Status code
GET for a workspace that does not exist	404
PUT that changes name of workspace	403
DELETE against a workspace that is non-empty	403

Parameters

recurse

The `recurse` parameter recursively deletes all layers referenced by the specified workspace, including data stores, coverage stores, feature types, and so on. Allowed values for this parameter are "true" or "false". The default value is "false".

quietOnNotFound

The `quietOnNotFound` parameter avoids to log an Exception when the Workspace is not present. Note that 404 status code will be returned anyway.

/workspaces/default[.<format>]

Controls the default workspace.

Method	Action	Status code	Formats	Default Format
GET	Returns default workspace	200	HTML, XML, JSON	HTML
POST		405		
PUT	200	Set default workspace	XML, JSON	
DELETE		405		

/workspaces/<ws>/settings[.<format>]

Controls settings on a specific workspace.

Method	Action	Status code	Formats	Default Format
GET	Returns workspace settings	200	HTML, XML, JSON	HTML
POST		405		
PUT	Creates or updates workspace settings	200	XML, JSON	
DELETE	Deletes workspace settings	200	XML, JSON	

Namespaces

A `namespace` is a uniquely identifiable grouping of feature types. It is identified by a prefix and a URI.

/namespaces[.<format>]

Controls all namespaces.

Method	Action	Status code	Formats	Default Format
GET	List all namespaces	200	HTML, XML, JSON	HTML
POST	Create a new namespace	201 with `Location` header	XML, JSON	
PUT		405		
DELETE		405		

/namespaces/<ns>[.<format>]

Controls a particular namespace.

Method	Action	Status code	Formats	Default Format	Parameters
GET	Return namespace `ns`	200	HTML, XML, JSON	HTML	*quietOnNotFound*
POST		405			
PUT	200	Modify namespace `ns`	XML, JSON		
DELETE	200	Delete namespace `ns`	XML, JSON		

Exceptions

Exception	Status code
GET for a namespace that does not exist	404
PUT that changes prefix of namespace	403
DELETE against a namespace whose corresponding workspace is non-empty	403

Parameters

quietOnNotFound

The `quietOnNotFound` parameter avoids to log an Exception when the Namespace is not present. Note that 404 status code will be returned anyway.

/namespaces/default[.<format>]

Controls the default namespace.

Method	Action	Status code	Formats	Default Format
GET	Return default namespace	200	HTML, XML, JSON	HTML
POST		405		
PUT	200	Set default namespace	XML, JSON	
DELETE		405		

Data stores

A `data store` contains vector format spatial data. It can be a file (such as a shapefile), a database (such as PostGIS), or a server (such as a *remote Web Feature Service*).

`/workspaces/<ws>/datastores[.<format>]`

Controls all data stores in a given workspace.

Method	Action	Status code	Formats	Default Format
GET	List all data stores in workspace `ws`	200	HTML, XML, JSON	HTML
POST	Create a new data store	201 with `Location` header	XML, JSON	
PUT		405		
DELETE		405		

`/workspaces/<ws>/datastores/<ds>[.<format>]`

Controls a particular data store in a given workspace.

Method	Action	Status code	Formats	Default Format	Parameters
GET	Return data store `ds`	200	HTML, XML, JSON	HTML	*quietOnNotFound*
POST		405			
PUT	Modify data store `ds`				
DELETE	Delete data store `ds`				*recurse*

Exceptions

Exception	Status code
GET for a data store that does not exist	404
PUT that changes name of data store	403
PUT that changes workspace of data store	403
DELETE against a data store that contains configured feature types	403

Parameters

recurse

The `recurse` parameter recursively deletes all layers referenced by the specified data store. Allowed values for this parameter are "true" or "false". The default value is "false".

quietOnNotFound

The `quietOnNotFound` parameter avoids to log an Exception when the data store is not present. Note that 404 status code will be returned anyway.

/workspaces/<ws>/datastores/<ds>/[file|url|external][.<extension>]

These endpoints (`file`, `url`, and `external`) allow a file containing either spatial data or a mapping configuration (in case an app-schema data store is targeted), to be added (via a PUT request) into an existing data store, or will create a new data store if it doesn't already exist. The three endpoints are used to specify the method that is used to upload the file:

- `file`—Uploads a file from a local source. The body of the request is the file itself.
- `url`—Uploads a file from an remote source. The body of the request is a URL pointing to the file to upload. This URL must be visible from the server.
- `external`—Uses an existing file on the server. The body of the request is the absolute path to the existing file.

Method	Action	Status code	Formats	Default Format	Parameters
GET	*Deprecated*. Retrieve the underlying files for the data store as a zip file with MIME type `application/zip`	200			
POST		405			
PUT	Uploads files to the data store `ds`, creating it if necessary	200	*See note below*		*configure*, *target*, *update*, *charset*
DELETE		405			

Exceptions

Exception	Status code
GET for a data store that does not exist	404
GET for a data store that is not file based	404

Parameters

extension

The `extension` parameter specifies the type of data being uploaded. The following extensions are supported:

Extension	Datastore
shp	Shapefile
properties	Property file
h2	H2 Database
spatialite	SpatiaLite Database
appschema	App-schema mapping configuration

Note: A file can be PUT to a data store as a standalone or zipped archive file. Standalone files are only suitable for data stores that work with a single file such as a GML store. Data stores that work with multiple files, such as the shapefile store, must be sent as a zip archive.

When uploading a standalone file, set the `Content-type` appropriately based on the file type. If you are loading a zip archive, set the `Content-type` to `application/zip`.

Note: The app-schema mapping configuration can either be uploaded as a single file, or split in multiple files for reusability and/or mapping constraints (e.g. multiple mappings of the same feature type are needed). If multiple mapping files are uploaded as a zip archive, the extension of the main mapping file (the one including the others via the `<includedTypes>` tag) must be `.appschema`, otherwise it will not be recognized as the data store's primary file and publishing will fail.

The application schemas (XSD files) required to define the mapping can be added to the zip archive and uploaded along with the mapping configuration. All files contained in the archive are uploaded to the same folder, so the path to the secondary mapping files and the application schemas, as specified in the main mapping file, is simply the file name of the included resource.

configure

The `configure` parameter controls how the data store is configured upon file upload. It can take one of the three values:

- `first`—(*Default*) Only setup the first feature type available in the data store.
- `none`—Do not configure any feature types.
- `all`—Configure all feature types.

Note: When uploading an app-schema mapping configuration, only the feature types mapped in the main mapping file are considered to be top level features and will be automatically configured when `configure=all` or `configure=first` is specified.

target

The `target` parameter determines what format or storage engine will be used when a new data store is created on the server for uploaded data. When importing data into an existing data store, it is ignored. The allowed values for this parameter are the same as for the *extension parameter*, except for `appschema`, which doesn't make sense in this context.

update

The `update` parameter controls how existing data is handled when the file is PUT into a data store that already exists and already contains a schema that matches the content of the file. The parameter accepts one of the following values:

- `append`—Data being uploaded is appended to the existing data. This is the default.
- `overwrite`—Data being uploaded replaces any existing data.

The parameter is ignored for app-schema data stores, which are read-only.

charset

The `charset` parameter specifies the character encoding of the file being uploaded (such as "ISO-8559-1").

Feature types

A `feature type` is a vector based spatial resource or data set that originates from a data store. In some cases, such as with a shapefile, a feature type has a one-to-one relationship with its data store. In other cases, such as PostGIS, the relationship of feature type to data store is many-to-one, feature types corresponding to a table in the database.

/workspaces/<ws>/datastores/<ds>/featuretypes[.<format>]

Controls all feature types in a given data store / workspace.

Method	Action	Status code	Formats	Default Format	Parameters
GET	List all feature types in data store `ds`	200	HTML, XML, JSON	HTML	*list*
POST	Create a new feature type, *see note below*	201 with `Location` header	XML, JSON		
PUT		405			
DELETE		405			

Note: When creating a new feature type via `POST`, if no underlying dataset with the specified name exists an attempt will be made to create it. This will work only in cases where the underlying data format supports the creation of new types (such as a database). When creating a feature type in this manner the client should include all attribute information in the feature type representation.

Exceptions

Exception	Status code
GET for a feature type that does not exist	404
PUT that changes name of feature type	403
PUT that changes data store of feature type	403

Parameters

`list`

The `list` parameter is used to control the category of feature types that are returned. It can take one of the following values:

- `configured`—Only configured feature types are returned. This is the default value.
- `available`—Only feature types that haven't been configured but are available from the specified data store will be returned.
- `available_with_geom`—Same as `available` but only includes feature types that have a geometry attribute.
- `all`—The union of `configured` and `available`.

`/workspaces/<ws>/datastores/<ds>/featuretypes/<ft>[.<format>]`

Controls a particular feature type in a given data store and workspace.

Method	Action	Status code	Formats	Default Format	Parameters
GET	Return feature type `ft`	200	HTML, XML, JSON	HTML	*quietOnNotFound*
POST		405			
PUT	Modify feature type `ft`	200	XML,JSON		*recalculate*
DELETE	Delete feature type `ft`	200			*recurse*

Exceptions

Exception	Status code
GET for a feature type that does not exist	404
PUT that changes name of feature type	403
PUT that changes data store of feature type	403

Parameters

recurse

The `recurse` parameter recursively deletes all layers referenced by the specified featuretype. Allowed values for this parameter are "true" or "false". The default value is "false". A DELETE request with `recurse=false` will fail if any layers reference the featuretype.

recalculate

The `recalculate` parameter specifies whether to recalculate any bounding boxes for a feature type. Some properties of feature types are automatically recalculated when necessary. In particular, the native bounding box is recalculated when the projection or projection policy are changed, and the lat/long bounding box is recalculated when the native bounding box is recalculated, or when a new native bounding box is explicitly provided in the request. (The native and lat/long bounding boxes are not automatically recalculated when they are explicitly included in the request.) In addition, the client may explicitly request a fixed set of fields to calculate, by including a comma-separated list of their names in the `recalculate` parameter. For example:

- `recalculate=` (empty parameter): Do not calculate any fields, regardless of the projection, projection policy, etc. This might be useful to avoid slow recalculation when operating against large datasets.
- `recalculate=nativebbox`: Recalculate the native bounding box, but do not recalculate the lat/long bounding box.
- `recalculate=nativebbox,latlonbbox`: Recalculate both the native bounding box and the lat/long bounding box.

quietOnNotFound

The `quietOnNotFound` parameter avoids to log an Exception when the feature type is not present. Note that 404 status code will be returned anyway.

Coverage stores

A `coverage store` contains raster format spatial data.

/workspaces/<ws>/coveragestores[.<format>]

Controls all coverage stores in a given workspace.

Method	Action	Status code	Formats	Default Format
GET	List all coverage stores in workspace `ws`	200	HTML, XML, JSON	HTML
POST	Create a new coverage store	201 with `Location` header	XML, JSON	
PUT		405		
DELETE		405		

/workspaces/<ws>/coveragestores/<cs>[.<format>]

Controls a particular coverage store in a given workspace.

Method	Action	Status code	Formats	Default Format	Parameters
GET	Return coverage store `cs`	200	HTML, XML, JSON	HTML	*quietOnNotFound*
POST		405			
PUT	Modify coverage store `cs`				
DELETE	Delete coverage store `cs`				*recurse*, *purge*

Exceptions

Exception	Status code
GET for a coverage store that does not exist	404
PUT that changes name of coverage store	403
PUT that changes workspace of coverage store	403
DELETE against a coverage store that contains configured coverage	403

Parameters

recurse

The `recurse` parameter recursively deletes all layers referenced by the coverage store. Allowed values for this parameter are "true" or "false". The default value is "false".

purge

The `purge` parameter is used to customize the delete of files on disk (in case the underlying reader implements a delete method). It can take one of the three values:

- `none`-(*Default*) Do not delete any store's file from disk.
- `metadata`-Delete only auxiliary files and metadata. It's recommended when data files (such as granules) should not be deleted from disk.
- `all`-Purge everything related to that store (metadata and granules).

quietOnNotFound

The `quietOnNotFound` parameter avoids to log an Exception when the coverage store is not present. Note that 404 status code will be returned anyway.

/workspaces/<ws>/coveragestores/<cs>/file[.<extension>]

This end point allows a file containing spatial data to be added (via a POST or PUT) into an existing coverage store, or will create a new coverage store if it doesn't already exist. In case of coverage stores containing multiple coverages (e.g., mosaic of NetCDF files) all the coverages will be configured unless `configure=false` is specified as a parameter.

Method	Action	Status code	Formats	Default Format	Parameters
GET	*Deprecated*. Get the underlying files for the coverage store as a zip file with MIME type `application/zip`.	200			
POST	If the coverage store is a simple one (e.g. GeoTiff) it will return a 405, if the coverage store is a structured one (e.g., mosaic) it will harvest the specified files into it, which in turn will integrate the files into the store. Harvest meaning is store dependent, for mosaic the new files will be added as new granules of the mosaic, and existing files will get their attribute updated, other stores might have a different behavior.	405 if the coverage store is a simple one, 200 if structured and the harvest operation succeeded			*recalculate*, *filename*
PUT	Creates or overwrites the files for coverage store `cs`	200	*See note below*		*configure*, *coverageName*
DELETE		405			

Note: A file can be PUT to a coverage store as a standalone or zipped archive file. Standalone files are only suitable for coverage stores that work with a single file such as GeoTIFF store. Coverage stores that work with multiple files, such as the ImageMosaic store, must be sent as a zip archive.

When uploading a standalone file, set the `Content-type` appropriately based on the file type. If you are loading a zip archive, set the `Content-type` to `application/zip`.

Exceptions

Exception	Status code
GET for a data store that does not exist	404
GET for a data store that is not file based	404

Parameters

extension

The `extension` parameter specifies the type of coverage store. The following extensions are supported:

Extension	Coverage store
geotiff	GeoTIFF
worldimage	Georeferenced image (JPEG, PNG, TIFF)
imagemosaic	Image mosaic

configure

The `configure` parameter controls how the coverage store is configured upon file upload. It can take one of the three values:

- `first`—(*Default*) Only setup the first feature type available in the coverage store.
- `none`—Do not configure any feature types.
- `all`—Configure all feature types.

coverageName

The `coverageName` parameter specifies the name of the coverage within the coverage store. This parameter is only relevant if the `configure` parameter is not equal to "none". If not specified the resulting coverage will receive the same name as its containing coverage store.

Note: At present a one-to-one relationship exists between a coverage store and a coverage. However, there are plans to support multidimensional coverages, so this parameter may change.

---

recalculate

The `recalculate` parameter specifies whether to recalculate any bounding boxes for a coverage. Some properties of coverages are automatically recalculated when necessary. In particular, the native bounding box is recalculated when the projection or projection policy is changed. The lat/long bounding box is recalculated when the native bounding box is recalculated or when a new native bounding box is explicitly provided in the request. (The native and lat/long bounding boxes are not automatically recalculated when they are explicitly included in the request.) In addition, the client may explicitly request a fixed set of fields to calculate by including a comma-separated list of their names in the `recalculate` parameter. For example:

- `recalculate=` (empty parameter)—Do not calculate any fields, regardless of the projection, projection policy, etc. This might be useful to avoid slow recalculation when operating against large datasets.
- `recalculate=nativebbox`—Recalculate the native bounding box, but do not recalculate the lat/long bounding box.
- `recalculate=nativebbox,latlonbbox`—Recalculate both the native bounding box and the lat/long bounding box.

filename

The `filename` parameter specifies the target file name for a file that needs to harvested as part of a mosaic. This is important to avoid clashes and to make sure the right dimension values are available in the name for multidimensional mosaics to work.

- `filename=`NCOM_wattemp_000_20081102T0000000_12.tiff` Set the uploaded file name to ``NCOM_wattemp_000_20081102T0000000_12.tiff`

Coverages

A `coverage` is a raster data set which originates from a coverage store.

/workspaces/<ws>/coveragestores/<cs>/coverages[.<format>]

Controls all coverages in a given coverage store and workspace.

Method	Action	Status code	Formats	Default Format
GET	List all coverages in coverage store `cs`	200	HTML, XML, JSON	HTML
POST	Create a new coverage	201 with `Location` header	XML, JSON	
PUT		405		
DELETE		405		

/workspaces/<ws>/coveragestores/<cs>/coverages/<c>[.<format>]

Controls a particular coverage in a given coverage store and workspace.

Method	Action	Status code	Formats	Default Format	Parameters
GET	Return coverage `c`	200	HTML, XML, JSON	HTML	*quietOnNotFound*
POST		405			
PUT	Modify coverage `c`	200	XML,JSON		
DELETE	Delete coverage `c`	200			*recurse*

Exceptions

Exception	Status code
GET for a coverage that does not exist	404
PUT that changes name of coverage	403
PUT that changes coverage store of coverage	403

Parameters

recurse

The `recurse` parameter recursively deletes all layers referenced by the specified coverage. Permitted values for this parameter are "true" or "false". The default value is "false".

quietOnNotFound

The `quietOnNotFound` parameter avoids to log an Exception when the coverage is not present. Note that 404 status code will be returned anyway.

Structured coverages

Structured coverages are the ones whose content is made of granules, normally associated to attributes, often used to represent time, elevation and other custom dimensions attached to the granules themselves. Image mosaic is an example of a writable structured coverage reader, in which each of the mosaic granules is associated with attributes. NetCDF is an example of a read only one, in which the multidimensional grid contained in the file is exposed as a set of 2D slices, each associated with a different set of variable values.

The following API applies exclusively to structured coverage readers.

/workspaces/<ws>/coveragestores/<cs>/coverages/<coverage>/index[.<format>]

Declares the set of attributes associated to the specified coverage, their name, type and min/max occurrences.

Method	Action	Status code	Formats	Default Format	Parameters
GET	Returns the attributes, their names and their types	200	XML, JSON	XML	
POST		405			
PUT		405			
DELETE		405			

/workspaces/<ws>/coveragestores/<cs>/coverages/<coverage>/index/granules.<format>

Returns the full list of granules, each with its attributes vales and geometry, and allows to selectively remove them

Method	Action	Status code	Formats	Default Format	Parameters
GET	Returns the list of granules and their attributes, either in GML (when XML is used) or GeoJSON (when JSON is used)	200	XML, JSON	XML	*offset*, *limit*, *filter*
POST		405			
PUT		405			
DELETE	Deletes the granules (all, or just the ones selected via the filter parameter)	200			*filter*

Parameters

`offset`

The `offset` parameter instructs GeoServer to skip the specified number of first granules when returning the data.

`limit`

The `limit` parameter instructs GeoServer to return at most the specified number of granules when returning the data.

`filter`

The `filter` parameter is a CQL filter that allows to select which granules will be returned based on their attribute values.

`/workspaces/<ws>/coveragestores/<cs>/coverages/<mosaic>/index/granules/<granuleId>.<format>`

Returns a single granule and allows for its removal.

Method	Action	Status code	Formats	Default Format	Parameters
GET	Returns the specified of granules and its attributes, either in GML (when XML is used) or GeoJSON (when JSON is used)	200	XML, JSON	XML	*quietOnNotFound*
POST		405			
PUT		405			
DELETE	Deletes the granule	200			

Exceptions

Exception	Status code
GET for a granule that does not exist	404

Parameters

`quietOnNotFound`

The `quietOnNotFound` parameter avoids to log an Exception when the granule is not present. Note that 404 status code will be returned anyway.

Styles

A `style` describes how a resource (feature type or coverage) should be symbolized or rendered by the Web Map Service. In GeoServer styles are specified with *SLD*.

`/styles[.<format>]`

Controls all styles.

Method	Action	Status code	Formats	Default Format	Parameters
GET	Return all styles	200	HTML, XML, JSON	HTML	
POST	Create a new style	201 with `Location` header	SLD, XML, JSON, ZIP *See note below*		*name raw*
PUT		405			
DELETE		405			

When executing a POST or PUT request with an SLD style, the `Content-type` header should be set to the mime type identifying the style format. Style formats supported out of the box include:

- SLD 1.0 with a mime type of `application/vnd.ogc.sld+xml`
- SLD 1.1 / SE 1.1 with a mime type of `application/vnd.ogc.se+xml`
- SLD package (zip file containing sld and image files used in the style) with a mime type of application/zip

Other extensions (such as *css*) add support for additional formats.

Parameters

`name`

The `name` parameter specifies the name to be given to the style. This option is most useful when executing a POST request with a style in SLD format, and an appropriate name can be not be inferred from the SLD itself.

raw

The `raw` parameter specifies whether to forgo parsing and encoding of the uploaded style content. When set to "true" the style payload will be streamed directly to GeoServer configuration. Use this setting if the content and formatting of the style is to be preserved exactly. Use this setting with care as it may result in an invalid and unusable style. The default is "false".

/styles/<s>[.<format>]

Controls a given style.

Method	Action	Status code	Formats	Default Format	Parameters
GET	Return style `s`	200	SLD, HTML, XML, JSON	HTML	*quietOnNotFound* *pretty*
POST		405			
PUT	Modify style `s`	200	SLD, XML, JSON, ZIP *See note above*		*raw*
DELETE	Delete style `s`	200		*purge*	*recurse*

Exceptions

Exception	Status code
GET for a style that does not exist	404
PUT that changes name of style	403
DELETE against style which is referenced by existing layers	403

Parameters

purge

The `purge` parameter specifies whether the underlying SLD file for the style should be deleted on disk. Allowable values for this parameter are "true" or "false". When set to "true" the underlying file will be deleted.

recurse

The `recurse` parameter removes references to the specified style in existing layers. Allowed values for this parameter are "true" or "false". The default value is "false".

quietOnNotFound

The `quietOnNotFound` parameter avoids to log an Exception when the style is not present. Note that 404 status code will be returned anyway.

pretty

The `pretty` parameter returns the style in a human-readable format, with proper whitespace and indentation. This parameter has no effect if you request a style in its native format - in this case the API returns the exact content of the underlying file. The HTML, XML, and JSON formats do not support this parameter.

/workspaces/<ws>/styles[.<format>]

Controls all styles in a given workspace.

Method	Action	Status code	Formats	Default Format	Parameters
GET	Return all styles within workspace `ws`	200	HTML, XML, JSON	HTML	
POST	Create a new style within workspace `ws`	201 with `Location` header	SLD, XML, JSON, ZIP *See note above*		*name raw*
PUT		405			
DELETE		405			*purge*

/workspaces/<ws>/styles/<s>[.<format>]

Controls a particular style in a given workspace.

Method	Action	Status code	Formats	Default Format	Parameters
GET	Return style `s` within workspace `ws`	200	SLD, HTML, XML, JSON	HTML	*quietOnNotFound*
POST		405			
PUT	Modify style `s` within workspace `ws`	200	SLD, XML, JSON, ZIP *See note above*		*raw*
DELETE	Delete style `s` within workspace `ws`	200			

Exceptions

Exception	Status code
GET for a style that does not exist for that workspace	404

Layers

A `layer` is a *published* resource (feature type or coverage).

/layers[.<format>]

Controls all layers.

Method	Action	Status code	Formats	Default Format
GET	Return all layers	200	HTML, XML, JSON	HTML
POST		405		
PUT		405		
DELETE		405		

/layers/<l>[.<format>]

Controls a particular layer.

Method	Action	Status code	Formats	Default Format	Parameters
GET	Return layer `l`	200	HTML, XML, JSON	HTML	*quietOnNotFound*
POST		405			
PUT	Modify layer `l`	200	XML,JSON		
DELETE	Delete layer `l`	200			*recurse*

Exceptions

Exception	Status code
GET for a layer that does not exist	404
PUT that changes name of layer	403
PUT that changes resource of layer	403

Parameters

recurse

The `recurse` parameter recursively deletes all styles referenced by the specified layer. Allowed values for this parameter are "true" or "false". The default value is "false".

quietOnNotFound

The `quietOnNotFound` parameter avoids to log an Exception when the layer is not present. Note that 404 status code will be returned anyway.

/layers/<l>/styles[.<format>]

Controls all styles in a given layer.

Method	Action	Status code	Formats	Default Format
GET	Return all styles for layer `l`	200	SLD, HTML, XML, JSON	HTML
POST	Add a new style to layer `l`	201, with `Location` header	XML, JSON	
PUT		405		
DELETE		405		

Layer groups

A `layer group` is a grouping of layers and styles that can be accessed as a single layer in a WMS GetMap request. A layer group is sometimes referred to as a "base map".

`/layergroups[.<format>]`

Controls all layer groups.

Method	Action	Status code	Formats	Default Format
GET	Return all layer groups	200	HTML, XML, JSON	HTML
POST	Add a new layer group	201, with `Location` header	XML,JSON	
PUT		405		
DELETE		405		

`/layergroups/<lg>[.<format>]`

Controls a particular layer group.

Method	Action	Status code	Formats	Default Format	Parameters
GET	Return layer group `lg`	200	HTML, XML, JSON	HTML	*quietOnNotFound*
POST		405			
PUT	Modify layer group `lg`	200	XML,JSON		
DELETE	Delete layer group `lg`	200			

Exceptions

Exception	Status code
GET for a layer group that does not exist	404
POST that specifies layer group with no layers	400
PUT that changes name of layer group	403

Parameters

quietOnNotFound

The `quietOnNotFound` parameter avoids to log an Exception when the layergroup is not present. Note that 404 status code will be returned anyway.

/workspaces/<ws>/layergroups[.<format>]

Controls all layer groups in a given workspace.

Method	Action	Status code	Formats	Default Format
GET	Return all layer groups within workspace `ws`	200	HTML, XML, JSON	HTML
POST	Add a new layer group within workspace `ws`	201, with `Location` header	XML,JSON	
PUT		405		
DELETE		405		

/workspaces/<ws>/layergroups/<lg>[.<format>]

Controls a particular layer group in a given workspace.

Method	Action	Status code	Formats	Default Format	
GET	Return layer group `lg` within workspace `ws`	200	HTML, XML, JSON	HTML	*quietOnNotFound*
POST		405			
PUT	Modify layer group `lg` within workspace `ws`	200	XML,JSON		
DELETE	Delete layer group `lg` within workspace `ws`	200			

Exceptions

Exception	Status code
GET for a layer group that does not exist for that workspace	404

Fonts

This operation provides the list of `fonts` available in GeoServer. It can be useful to use this operation to verify if a `font` used in a SLD file is available before uploading the SLD.

/fonts[.<format>]

Method	Action	Status code	Formats	Default Format
GET	Return the fonts available in GeoServer	200	XML, JSON	XML
POST		405		
PUT		405		
DELETE		405		

Freemarker templates

Freemarker is a simple yet powerful template engine that GeoServer uses for user customization of outputs. It is possible to use the GeoServer REST API to manage Freemarker templates for catalog resources.

/templates/<template>.ftl

This endpoint manages a template that is global to the entire catalog.

Method	Action	Status Code	Formats	Default Format
GET	Return a template	200		
PUT	Insert or update a template	405		
DELETE	Delete a template	405		

Identical operations apply to the following endpoints:

- Workspace templates—`/workspaces/<ws>/templates/<template>.ftl`
- Vector store templates—`/workspaces/<ws>/datastores/<ds>/templates/<template>.ftl`
- Feature type templates—`/workspaces/<ws>/datastores/<ds>/featuretypes/<f>/templates/<template>.ftl`
- Raster store templates—`/workspaces/<ws>/coveragestores/<cs>/templates/<template>.ftl`
- Coverage templates—`/workspaces/<ws>/coveragestores/<cs>/coverages/<c>/templates/<template>.ftl`

/templates[.<format>]

This endpoint manages all global templates.

Method	Action	Status Code	Formats	Default Format
GET	Return templates	200	HTML, XML, JSON	HTML

Identical operations apply to the following endpoints:

- Workspace templates—`/workspaces/<ws>/templates[.<format>]`
- Vector store templates—`/workspaces/<ws>/datastores/<ds>/templates[.<format>]`

- Feature type templates—`/workspaces/<ws>/datastores/<ds>/featuretypes/<f>/templates[.<format>]`
- Raster store templates—`/workspaces/<ws>/coveragestores/<cs>/templates[.<format>]`
- Coverage templates—`/workspaces/<ws>/coveragestores/<cs>/coverages/<c>/templates[.<format>]`

OWS Services

GeoServer includes several types of OGC services like WCS, WFS and WMS, commonly referred to as "OWS" services. These services can be global for the whole GeoServer instance or local to a particular workspace. In this last case, they are called *virtual services*.

`/services/wcs/settings[.<format>]`

Controls Web Coverage Service settings.

Method	Action	Status code	Formats	Default Format
GET	Return global WCS settings	200	XML, JSON	HTML
POST		405		
PUT	Modify global WCS settings	200		
DELETE		405		

`/services/wcs/workspaces/<ws>/settings[.<format>]`

Controls Web Coverage Service settings for a given workspace.

Method	Action	Status code	Formats	Default Format
GET	Return WCS settings for workspace `ws`	200	HTML, XML, JSON	HTML
POST		405		
PUT	Create or modify WCS settings for workspace `ws`	200	XML,JSON	
DELETE	Delete WCS settings for workspace `ws`	200		

`/services/wfs/settings[.<format>]`

Controls Web Feature Service settings.

Method	Action	Status code	Formats	Default Format
GET	Return global WFS settings	200	HTML, XML, JSON	HTML
POST		405		
PUT	Modify global WFS settings	200	XML,JSON	
DELETE		405		

/services/wfs/workspaces/<ws>/settings[.<format>]

Controls Web Feature Service settings for a given workspace.

Method	Action	Status code	Formats	Default Format
GET	Return WFS settings for workspace `ws`	200	HTML, XML, JSON	HTML
POST		405		
PUT	Modify WFS settings for workspace `ws`	200	XML,JSON	
DELETE	Delete WFS settings for workspace `ws`	200		

/services/wms/settings[.<format>]

Controls Web Map Service settings.

Method	Action	Status code	Formats	Default Format
GET	Return global WMS settings	200	HTML, XML, JSON	HTML
POST		405		
PUT	Modify global WMS settings	200	XML,JSON	
DELETE		405		

/services/wms/workspaces/<ws>/settings[.<format>]

Controls Web Map Service settings for a given workspace.

Method	Action	Status code	Formats	Default Format
GET	Return WMS settings for workspace `ws`	200	HTML, XML, JSON	HTML
POST		405		
PUT	Modify WMS settings for workspace `ws`	200	XML,JSON	
DELETE	Delete WMS settings for workspace `ws`	200		

Reloading configuration

Reloads the GeoServer catalog and configuration from disk. This operation is used in cases where an external tool has modified the on-disk configuration. This operation will also force GeoServer to drop any internal caches and reconnect to all data stores.

/reload

Method	Action	Status code	Formats	Default Format
GET		405		
POST	Reload the configuration from disk	200		
PUT	Reload the configuration from disk	200		
DELETE		405		

Resource reset

Resets all store, raster, and schema caches. This operation is used to force GeoServer to drop all caches and store connections and reconnect to each of them the next time they are needed by a request. This is useful in case the stores themselves cache some information about the data structures they manage that may have changed in the meantime.

/reset

Method	Action	Status code	Formats	Default Format
GET		405		
POST	Reset all store, raster, and schema caches	200		
PUT	Reset all store, raster, and schema caches	200		
DELETE		405		

Manifests

GeoServer provides a REST service to expose a listing of all loaded JARs and resources on the running instance. This is useful for bug reports and to keep track of extensions deployed into the application. There are two endpoints for accessing this information:

- `about/manifest`—Retrieves details on all loaded JARs
- `about/version`—Retrieves details for the high-level components: GeoSever, GeoTools, and GeoWebCache
- `about/status`-Retrieves details for the status of all loaded and configured modules

/about/manifest[.<format>]

This endpoint retrieves details on all loaded JARs.

All the GeoServer manifest JARs are marked with the property `GeoServerModule` and classified by type, so you can use filtering capabilities to search for a set of manifests using regular expressions (see the *manifest* parameter) or a type category (see the *key* and *value* parameter).

The available types are `core`, `extension`, or `community`. To filter modules by a particular type, append a request with `key=GeoServerModule&value=<type>`

Method	Action	Status Code	Formats	Default Format	Parameters
GET	List all manifests into the classpath	200	HTML, XML, JSON	HTML	*manifest*, *key*, *value*
POST		405			
PUT		405			
DELETE		405			

Usage

The model is very simple and is shared between the version and the resource requests to parse both requests.:

```
<about>
  <resource name="{NAME}">
    <{KEY}>{VALUE}</{KEY}>
    ...
  </resource>
  ...
</about>
```

You can customize the results adding a properties file called `manifest.properties` into the root of the data directory. Below is the default implementation that is used when no custom properties file is present:

```
resourceNameRegex=.+/(.*).(jar|war)
resourceAttributeExclusions=Import-Package,Export-Package,Class-Path,Require-Bundle
versionAttributeInclusions=Project-Version:Version,Build-Timestamp,Git-Revision,
  Specification-Version:Version,Implementation-Version:Git-Revision
```

where:

- `resourceNameRegex`—Group(1) will be used to match the attribute name of the resource.
- `resourceAttributeExclusions`—Comma-separated list of properties to exclude (blacklist), used to exclude parameters that are too verbose such that the resource properties list is left open. Users can add their JARs (with custom properties) having the complete list of properties.
- `versionAttributeInclusions`—Comma-separated list of properties to include (whitelist). Also supports renaming properties (using `key:replace`) which is used to align the output of the `versions` request to the output of the web page. The model uses a map to store attributes, so the last attribute found in the manifest file will be used.

manifest

The `manifest` parameter is used to filter over resulting resource (manifest) names attribute using Java regular expressions.

key

The `key` parameter is used to filter over resulting resource (manifest) properties name. It can be combined with the `value` parameter.

value

The `value` parameter is used to filter over resulting resource (manifest) properties value. It can be combined with the `key` parameter.

/about/version[.<format>]

This endpoint shows only the details for the high-level components: GeoServer, GeoTools, and GeoWebCache.

Method	Action	Status Code	Formats	Default Format	Parameters
GET	List GeoServer, GeoWebCache and GeoTools manifests	200	HTML, XML, JSON	HTML	*manifest*, *key*, *value*
POST		405			
PUT		405			
DELETE		405			

/about/status[.<format>]

This endpoint shows the status details of all installed and configured modules.Status details always include human readable name, and module name. Optional details include version, availability, status message, and links to documentation.

Method	Action	Status Code	Formats	Default Format	Parameters
GET	List module statuses	200	HTML, XML, JSON	HTML	
POST		405			
PUT		405			
DELETE		405			

Master Password

The `master password` is used to encrypt the GeoServer key store and for an emergency login using the user `root`.

Warning: The use of HTTPS is recommended, otherwise all password are sent in plain text.

/security/masterpw[.<format>]

Fetches the master password and allows to change the master password

Method	Action	Status code	Formats	Default Format
GET	Fetch the master password	200,403	XML, JSON	
PUT	Changes the master password	200,405,422	XML, JSON	

Formats for PUT (master password change).

XML

```
<masterPassword>
   <oldMasterPassword>oldPassword</oldMasterPassword>
   <newMasterPassword>newPassword</newMasterPassword>
</masterPassword>
```

JSON

```
{ "oldMasterPassword":"oldPassword",
  "newMasterPassword":"newPassword" }
```

Exceptions

Exception	Status code
GET without administrative privileges	403
PUT without administrative privileges	405
PUT with the wrong current master password	422
PUT with a new master password rejected by the password policy	422

Self admin

Self admin operations allow a user to perform actions on the user's own info.

Calls to the self admin operations are disabled by default. You'll have to edit the `rest.properties` file (more info at the *REST services* page) and add the line:

```
/rest/security/self/**;GET,POST,PUT,DELETE=ROLE_AUTHENTICATED
```

/security/self/password

Allows a user to change own password

> **Warning:** The use of HTTPS is recommended, otherwise all password are sent in plain text.

Method	Action	Status code	Formats	Default Format
PUT	Changes the user password	200,400,424	XML, JSON	

Formats for PUT (password change).

XML

```
<userPassword>
   <newPassword>newPassword</newPassword>
</userPassword>
```

JSON

```
{ "newPassword":"newPassword" }
```

Exceptions

Exception	Status code	Error string (payload)
PUT with an invalid `newPassword` or bad params	400	`Missing 'newPassword'`
PUT for user not updatable	424	`User service does not support changing pw`

Access Control

`/security/acl/catalog.<format>`

Fetches the catalog mode and allows to change the catalog mode. The mode must be one of

- HIDE
- MIXED
- CHALLENGE

Method	Action	Status code	Formats	Default Format
GET	Fetch the catalog mode	200,403	XML, JSON	
PUT	Set the catalog mode	200,403,404,422	XML, JSON	

Formats:

XML

```
<catalog>
  <mode>HIDE</mode>
</catalog>
```

JSON

```
{"mode":"HIDE" }
```

Exceptions

Exception	Status code
No administrative privileges	403
Malformed request	404
Invalid catalog mode	422

/security/acl/layers.<format>

/security/acl/services.<format>

/security/acl/rest.<format>

API for administering access control for

- Layers
- Services
- The REST API

Method	Action	Status code	Formats	Default Format
GET	Fetch all rules	200,403	XML, JSON	
POST	Add a set of rules	200,403,409	XML, JSON	
PUT	Modify a set of rules	200,403,409	XML, JSON	
DELETE	Delete a specific rule	200,404,409	XML, JSON	

Format for DELETE:

The specified rule has to be the last part in the URI:

```
/security/acl/layers/*.*.r
```

Note: Slashes ("/") in a rule name must be encoded with **%2F**. The REST rule **/**;GET** must be encoded to /security/acl/rest/**%2F**;GET**

Formats for GET,POST and PUT:

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<rules>
   <rule resource="*.*.r">*</rule>
   <rule resource="myworkspace.*.w">ROLE_1,ROLE_2</rule>
</rules>
```

JSON

```
{
"*.*.r": "*",
"myworkspace".*.w": "ROLE_1,ROLE_2"
}
```

The resource attribute specifies a rule. There are three different formats.

- For layers: <workspace>.<layer>.<access>. The asterisk is a wild card for <workspace> and <layer>. <access> is one of **r** (read), **w** (write) or **a** (administer).
- For services: <service>.<method>. The asterisk is a wild card wild card for <service> and <method>. Examples:
  - wfs.GetFeature
  - wfs.GetTransaction

  - wfs.*
- For REST: <URL Ant pattern>;<comma separated list of HTTP methods>. Examples:
  - /**;GET
  - /**;POST,DELETE,PUT

The content of a rule element is a comma separated list of roles or the asterisk.

Exceptions

Exception	Status code
No administrative privileges	403
POST, adding an already existing rule	409
PUT, modifying a non existing rule	409
DELETE, Deleting a non existing rule	409
Invalid rule specification	422

Note: When adding a set of rules and only one role does already exist, the whole request is aborted. When modifying a set of rules and only one role does not exist, the whole request is aborted too.

Users/Groups and Roles

Security

The Users/Groups and Roles Rest API is only accessible to users with the role ROLE_ADMIN.

Input/Output

Data Object Transfer

Both XML and JSON are supported for transfer of data objects. The default is XML. Alternatively, JSON may be used by setting the 'content-type' (POST) and 'accept' (GET) http headers to 'application/json' in your requests.

Encoding of a user in XML:

```
<user>
        <userName>..</userName>
        <password>..</password>
        <enabled>true/false</enabled>
</user>
```

Encoding of a user in JSON:

```
{"userName": "..", "password": "..", enabled: true/false}
```

Passwords are left out in results of reading requests.

Encoding of a list of users in XML:

```
<users>
        <user> ... </user>
        <user> ... </user>
        ...
</users>
```

Encoding of a list of users in JSON:

```
{"users":[ {..}, {..}, .. ]}
```

Encoding of a list of groups in XML:

```
<groups>
        <group> agroupname </group>
        <group> bgroupname </group>
        ...
</groups>
```

Encoding of a list of groups in JSON:

```
{"groups":[ {..}, {..}, .. ]}
```

Encoding of a list of roles:

```
<roles>
        <role> arolename </role>
        <role> brolename </role>
        ...
</roles>
```

Encoding of a list of roles in JSON:

```
{"roles":[ {..}, {..}, .. ]}
```

Configuration

The default user/group service is by default the service named "default". This can be altered in the following manner:

1. Start geoserver with the following java system property present:

   ```
   org.geoserver.rest.DefaultUserGroupServiceName=<name_of_usergroupservice>
   ```

Requests

`/rest/usergroup/[service/<serviceName>/]users/`

Query all users or add a new user in a particular or the default user/group service.

Method	Action	Response
GET	List all users in service.	200 OK. List of users in XML.
POST	Add a new user	201 Inserted. Created `ID` header.

/rest/usergroup/[service/<serviceName>/]user/<user>

Query, modify or delete a specific user in a particular or the default user/group service.

Method	Action	Response
GET	Read user information	200 OK. User in XML.
POST	Modify the user, unspecified fields remain unchanged.	200 OK.
DELETE	Delete the user	200 OK.

/rest/usergroup/[service/<serviceName>/]groups/

Query all groups in a particular user/group or the default service.

Method	Action	Response
GET	List all groups in service.	200 OK. List of groups in XML.

/rest/usergroup/[service/<serviceName>/]group/<group>

Add or delete a specific group in a particular or the default user/group service.

Method	Action	Response
POST	Add the group.	200 OK.
DELETE	Delete the group.	200 OK.

/rest/usergroup/[service/<serviceName>/]user/<user>/groups

Query all groups associated with a user in a particular or the default user/group service.

Method	Action	Response
GET	List all groups associated with user.	200 OK. List of groups in XML.

/rest/usergroup/[service/<serviceName>/]group/<group>/users

Query all users associated with a group in a particular or the default user/group service.

Method	Action	Response
GET	List all users associated with group.	200 OK. List of groups in XML.

/rest/usergroup/[service/<serviceName>/]<user>/group/<group>

Associate or disassociate a specific user with a specific group in a particular or the default user/group service.

Method	Action	Response
POST	Associate the user with the group.	200 OK.
DELETE	Disassociate the user from the group.	200 OK.

rest/roles/[service/{serviceName}/]

Query all roles in a particular role service or the active role service.

Method	Action	Response
GET	List all roles in service.	200 OK. List of roles in XML.

/rest/roles/[service/<serviceName>/]role/<role>

Add or delete a specific role in a particular role service or the active role service.

Method	Action	Response
POST	Add the role.	200 OK.
DELETE	Delete the role.	200 OK.

/rest/roles/[service/<serviceName>/]<serviceName>/user/<user>/roles

Query all roles associated with a user in a particular role service or the active role service.

Method	Action	Response
GET	List all roles associated with user.	200 OK. List of roles in XML.

/rest/roles/[service/<serviceName>/]role/<role>/user/<user>/

Associate or disassociate a specific user with a specific role in a particular role service or the active role service.

Method	Action	Response
POST	Associate the user with the role.	200 OK.
DELETE	Disassociate the user from the role.	200 OK.

Resources

/resource</path/to/resource>

Method	Action	Status code	Parameters
GET	Download a resource, list contents of directory, or show formatted resource metadata.	200	operation (default | metadata); format (html | xml | json)
HEAD	Show resource metadata in HTTP headers.	200	
PUT	Upload/move/copy a resource, create directories on the fly (overwrite if exists). For move/copy operations, place source path in body. Copying is not supported for directories.	200 (exists) 201 (new)	operation (default | copy | move)
DELETE	Delete a resource (recursively if directory)	200	

Exceptions

Exception	Status code
GET or DELETE for a resource that does not exist	404
PUT to directory	405
PUT method=copy with source directory	405
PUT with source path that doesn't exist	404
POST	405

Headers

Header	Description
Last-Modified	When resource was last modified.
Content-Type	Will guess mime-type from extension or content.
Resource-Type (custom)	directory | resource
Resource-Parent (custom)	Path to parent

Format

Examples are given in XML. The JSON and HTML formats are analogue.

Metadata

```
<ResourceMetaData>
   <name> nameOfFile </name>
   <parent> <path> path/to/parent </path>
        <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate"
   href="http://localhost:8080/geoserver/rest/resource/path/to/parent?
↪operation=metadata&format=xml"
         type="application/xml"/>
```

```
   </parent>
   <lastModified> date </lastModified>
   <type> undefined | resource | directory </type>
</ResourceMetaData>
```

Directories

```
<ResourceDirectory>`
   <name> nameOfDirectory </name>
   <parent> <path> path/to/parent </path>
        <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate"
   href="http://localhost:8080/geoserver/rest/resource/path/to/parent?
↪operation=metadata&format=xml"
         type="application/xml"/>
   </parent>
   <lastModified> date </lastModified>
   <children>
       <child>
               <name> ... </name>
               <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate"
                href="http://localhost:8080/geoserver/rest/resource/path/to/child"/>
       </child>
       <child>
               ...
       </child>
       ...
   </children>`
</ResourceDirectory>
```

CHAPTER 13

Security

This section details the security subsystem in GeoServer, which is based on Spring Security.

The first page discusses configuration options in the web administration interface. This is followed with a detailed discussion of the underlying processes.

13.1 Security settings

GeoServer has a robust *security subsystem*, modeled on Spring Security. Most of the security features are available through the *Web administration interface*. This section describes how to configure GeoServer security through the web administration interface.

13.1.1 Settings

The Settings page controls the global GeoServer security settings.

Active role service

This option sets the active *role service* (provides information about roles). Role services are managed on the *Users, Groups, Roles* page. There can be only one active role service at one time.

Encryption

The GeoServer user interface (UI) can sometimes expose parameters in plain text inside the URLs. As a result, it may be desirable to encrypt the URL parameters. To enable encryption, select *Encrypt web admin URL parameters*. This will configure GeoServer to uses a PBE-based *Password encryption*.

For example, with this feature enabled, the page:

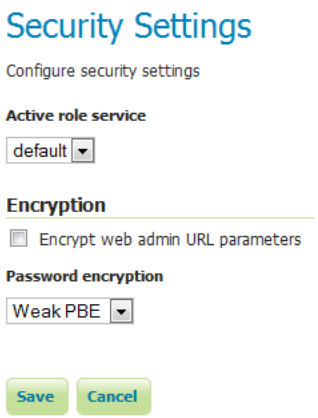


Fig. 13.1: Security Settings page

```
http://GEOSERVER/web/?wicket:bookmarkablePage=:org.geoserver.security.web.
↪SecuritySettingsPage
```

would now be found at the following URL:

```
http://GEOSERVER/web/?
↪x=hrTNYMcF3OY7u4NdyYnRanL6a1PxMdLxTZcY5xK5ZXyi617EFEFCagMwHBWhrlg*ujTOyd17DLSn0NO2JKO1Dw
```

Password encryption

This setting allows you to select the type of *Password encryption* used for passwords. The options are *Plain text*, *Weak PBE*, or *Strong PBE*.

If Strong PBE is not available as part of the JVM, a warning will display and the option will be disabled. To enable Strong PBE, you must install external policy JARs that support this form of encryption. See the section on *Password encryption* for more details about these settings.

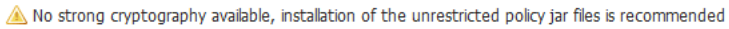


Fig. 13.2: Warning if Strong PBE is not available

13.1.2 Authentication

This page manages the authentication options, including authentication providers and the authentication chain.

Brute force attack prevention

GeoServer ships with a delay based brute force attack prevention system.

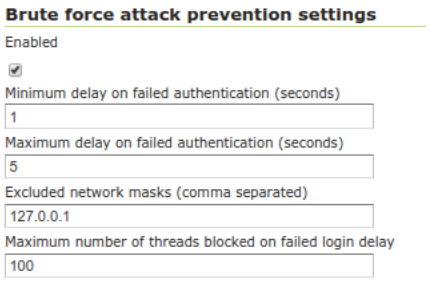


Fig. 13.3: *Brute force attack prevention settings*

Option	Description
Enabled	Whether the brute force attack prevention is enabled. Defaults to true.
Minimum delay on failed authentication (seconds)	Minimum number of seconds a failed login request will be made to wait before getting a response
Maximum delay on failed authentication (seconds)	Maximum number of seconds a failed login request will be made to wait before getting a response
Excluded network masks	Network masks identifying hosts that are excluded from brute force attack prevention. Can be empty, include specific IPs, or a list of network masks. Defaults to 127.0.0.1, the localhost.
Maximum number of threads blocked on failed login delay	Limits the number of threads that get delayed on failed login, should be set to a value less than the container's available response threads.

The mechanism works as follows:

- Each failed authentication request is made to wait between min and max seconds before getting an actual response back
- Each attempt to authenticate the same username in parallel fails immediately, regardless of whether the credentials were valid or not, with a message stating concurrent loging attempts are not allowed.

The first item slows down a single threaded attack to the point of making it ineffective (each failed attempt is logged along with the IP attempting access), the second item breaks multi-threaded attacks ability to scale. Login attempts are slowed down/blocked on all protocols, be either a OGC request, a REST call, or the UI.

A user trying to login from the user interface while another request is blocked waiting for the cool-down period to expire will see a message like the following:

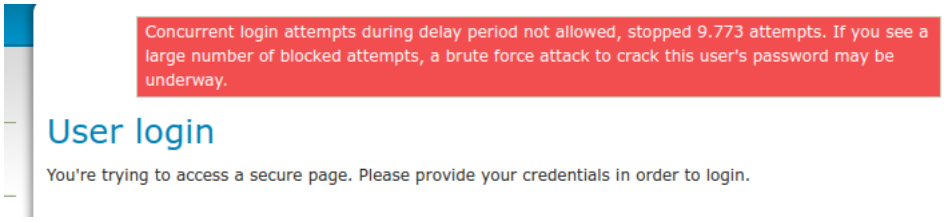


Fig. 13.4: *Error message for parallel user interface login*

A HTTP request (REST or OGC) will instead get an immediate 401 with a message like:

HTTP/1.1 401 User foo, 5896 concurrent login attempt(s) denied during the quiet period

A blessed set of IPs that can dodge the mechanism allows legit administrators to take control of the server even during an attack. The system only trusts the actual requestor IP, ignoring "X-Forwarded-For" headers, as they can be easily spoofed (this in turn requires the admin to access the system from a local network, without proxies in the middle, for the blessed IP to be recognized).

The maximum number of threads blocked configuration allows to setup the system so that an attacker can mis-use the system to simply block all service threads, by issuing requests with random user names (the system cannot determine if a username is valid or not, none of the authentication mechanisms provides this information for security reasons).

Considerations on how to setup the system:

- A small delay is normally more than enough to stop a brute force attack, resist the temptation of setting high delay values as they might end up blocking too many legit account and trigger the max blocked threads mechanism
- Ensure that the execluded networks are well protected by other means
- Set the maximum number of blocked threads to a value large allow peak hour legit logins (e.g., early morning when all the users start working) while still leaving room for successful authentication requests
- A clustered/load balanced setup will not share the state of blocked logins, each host tracks its local login failures.

Authentication filters

This section manages the Authentication Filters (adding, removing, and editing). Several authentication filters are configured by default (anonymous, basic, form, rememberme), but others can be added to the list.

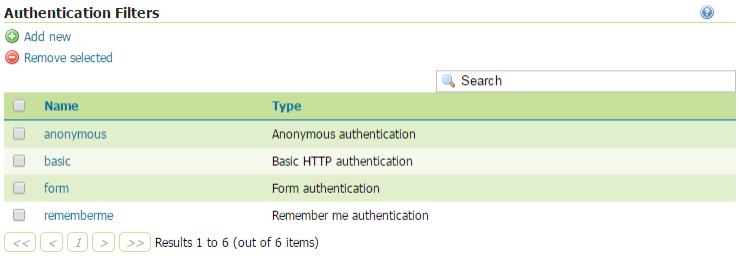


Fig. 13.5: *List of authentication filters*

Anonymous access

By default, GeoServer will allow anonymous access to the *Web administration interface*. Without authentication, users will still be able to view the *Layer Preview*, capabilities documents, and basic GeoServer details. Anonymous access can by removing the *anonymous* authentication filter. If removed, anonymous users navigating to the GeoServer page will get an HTTP 401 status code, which typically results in a browser-based request for credentials.

Credentials From Headers filter

This filter allows gathering user credentials (username and password) from request headers in a flexible and configurable way.

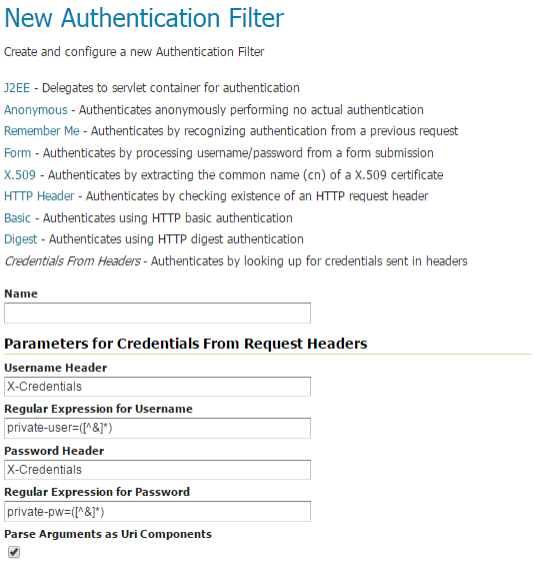


Fig. 13.6: *Creating a new authentication filter fetching credentials from request headers*

Option	Description
Name	Name of the filter
Username Header	Name of the Request Header containing the username
Regular Expression for Username	Regular Expression used to extract the username from the related Header. Must define a group that will match the username.
Password Header	Name of the Request Header containing the password
Regular Expression for Password	Regular Expression used to extract the password from the related Header. Must define a group that will match the password.
Parse Arguments as Uri Components	If checked username and password are uri deocded before being used as credentials

Authentication providers

This section manages the *Authentication providers* (adding, removing, and editing). The default authentication provider uses basic *username/password authentication*. *JDBC* and *LDAP* authentication can also be used.

Click *Add new* to create a new provider. Click an existing provider to edit its parameters.

Username/password provider

The default new authentication provider uses a user/group service for authentication.

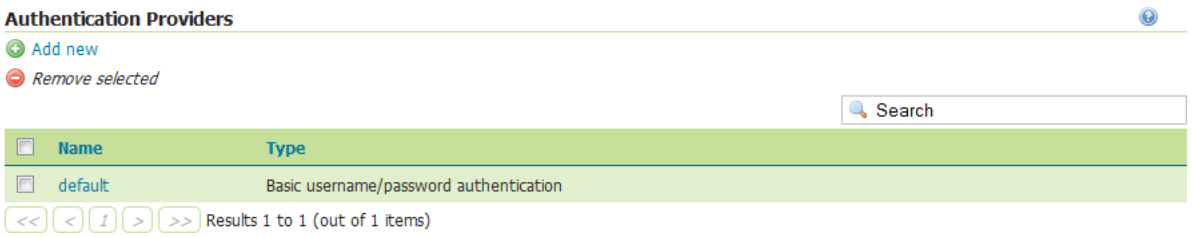


Fig. 13.7: *List of authentication providers*

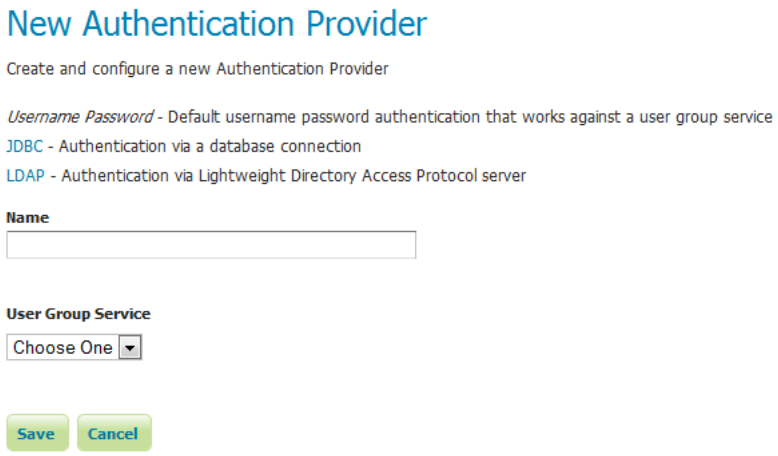


Fig. 13.8: *Creating a new authentication provider with a username and password*

Option	Description
Name	Name of the provider
User Group Service	Name of the user/group service associated with this provider. Can be any one of the active user/group services.

JDBC provider

The configuration options for the JDBC authentication provider are illustrated below.

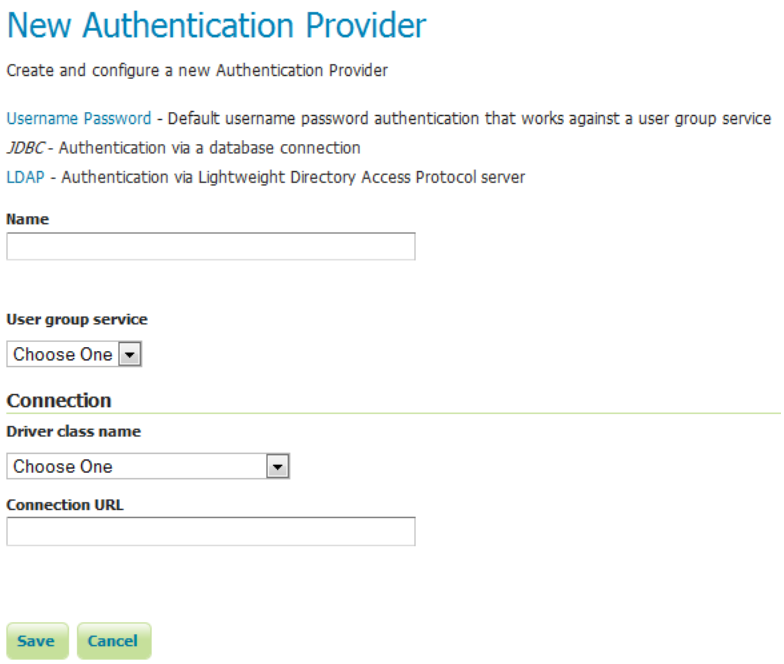


Fig. 13.9: *Configuring the JDBC authentication provider*

Option	Description
Name	Name of the JDBC connection in GeoServer
User Group Service	Name of the user/group service to use to load user information after the user is authenticated
Driver class name	JDBC driver to use for the database connection
Connection URL	JDBC URL to use when creating the database connection

LDAP provider

The following illustration shows the configuration options for the LDAP authentication provider. The default option is to use LDAP groups for role assignment, but there is also an option to use a user/group service for role assignment. Depending on whether this option is selected, the page itself will have different options.

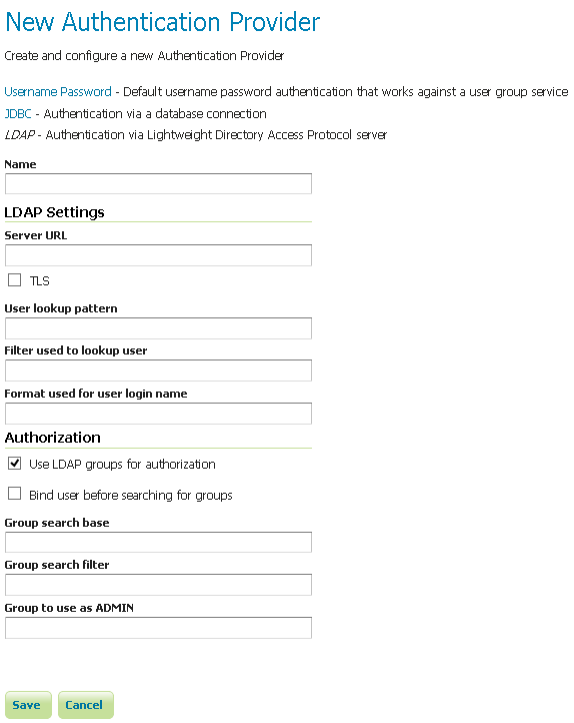


Fig. 13.10: *Configuring the LDAP authentication provider using LDAP groups for role assignment*

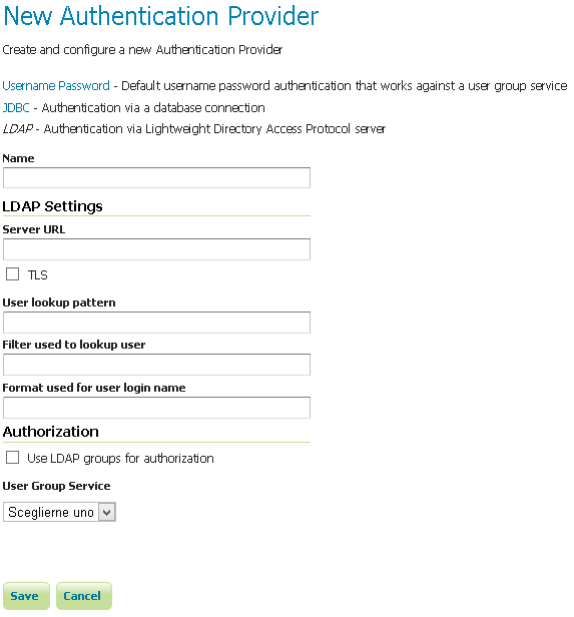


Fig. 13.11: *Configuring the LDAP authentication provider using user/group service for authentication*

Option	Description
Name	Name of the LDAP connection in GeoServer
Server URL	URL for the LDAP server connection. It must include the protocol, host, and port, as well as the "distinguished name" (DN) for the root of the LDAP tree.
TLS	Enables a STARTTLS connection. (See the section on *Secure LDAP connections*.)
User DN pattern	Search pattern to use to match the DN of the user in the LDAP database. The pattern should contain the placeholder `{0}` which is injected with the `uid` of the user. Example: `uid={0},ou=people`. The root DN specified as port of the *Server URL* is automatically appended.
User Filter	LDAP Filter used to extract User data from LDAP database. Used alternatively to User DN pattern and combined with User Format to separate bind and user data extraction handling. Example: `(userPrincipalName={0})`. Gets user data searching for a single record matching the filter. This may contain two placeholder values: `{0}`, the full DN of the user, for example `uid=bob,ou=people,dc=acme,dc=com` `{1}`, the `uid` portion of the full DN, for example `bob`.
User Format	String formatter used to build username used for binding. Used alternatively to User DN pattern and combined with User Filter to separate bind and user data extraction handling. Example: `{0}@domain`. Binds user with the username built applying the format. This may contain one placeholder: `{0}`, the username, for example `bob`
Use LDAP groups for authorization	Specifies whether to use LDAP groups for role assignment
Bind before group search	Specifies whether to bind to LDAP server with the user credentials before doing group search
Group search base	Relative name of the node in the tree to use as the base for LDAP groups. Example: `ou=groups`. The root DN specified as port of the *Server URL* is automatically appended. Only applicable when the *Use LDAP groups for authorization(parameter is **checked**.*
Group search filter	Search pattern for locating the LDAP groups a user belongs to. This may contain two placeholder values: `{0}`, the full DN of the user, for example `uid=bob,ou=people,dc=acme,dc=com` `{1}`, the `uid` portion of the full DN, for example `bob`. Only applicable when the *Use LDAP groups for authorization(parameter is **checked**.*
Admin Group	Name of the group to be mapped to Administrator role (defaults to ADMINISTRATOR). Example: `ADMIN`. Adds the role ROLE_ADMINISTRATOR if the user belongs to a group named ADMIN (case insensitive)
Group Admin Group	Name of the group to be mapped to Group Administrator role (defaults to GROUP_ADMIN). Example: `GROUPADMIN`. Adds the role ROLE_GROUP_ADMIN if the user belongs to a group named GROUPADMIN (case insensitive)
User Group Service	The user/group service to use for role assignment. Only applicable when the *Use LDAP groups for authorization* parameter is **cleared**.

Authentication chain

This section selects the authentication chain. Currently, only one default authentication chain is available. For further information about the default chain, please refer to *Authentication chain*.

Fig. 13.12: *Selecting the authentication chain*

13.1.3 Passwords

This page configures the various options related to *Passwords*, the *Master password*, and *Password policies*.

Note: User passwords may be changed in the Users dialog box accessed from the *Users, Groups, Roles* page.

Active master password provider

This option sets the active master password provider, via a list of all available master password providers.

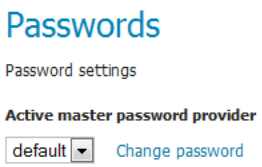


Fig. 13.13: Active master password provider

To change the master password click the *Change password* link.

Warning: First thing to do as an Administrator of the System, would be to dump the Master Password generated by GeoServer, store it somewhere not accessible by anyone, and delete any `security/masterpw.info` or whatever file you used to dump the password in clear.

Master Password Providers

This section provides the options for adding, removing, and editing master password providers.

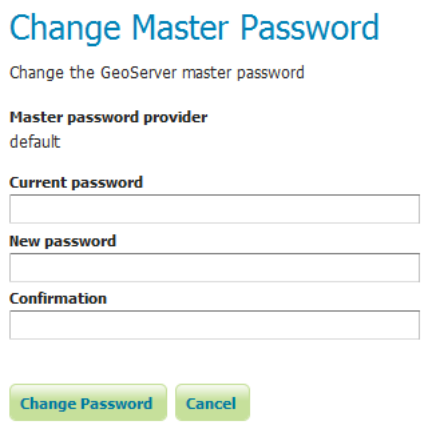


Fig. 13.14: Changing the master password

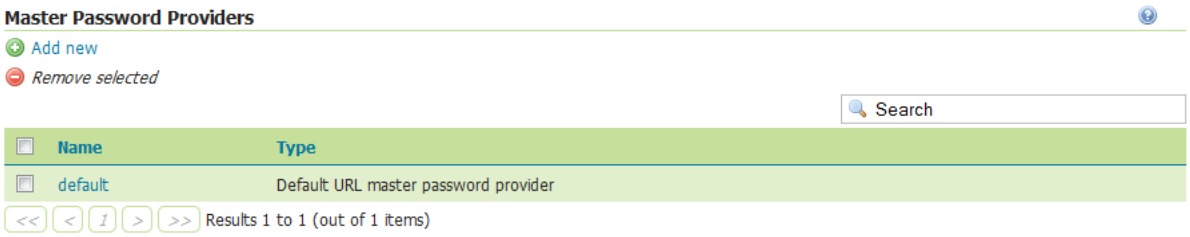


Fig. 13.15: Master password provider list

Note: By default the login to Admin GUI and REST APIs with Master Password is disabled. In order to enable it you will need to manually change the Master Password Provider `config.xml`, usually located into `security/masterpw/default/config.xml`, by adding the following statement:

```
``<loginEnabled>true</loginEnabled>``
```

Password policies

This section configures the various *Password policies* available to users in GeoServer. New password policies can be added or renamed, and existing policies edited or removed.

By default there are two password policies in effect, `default` and `master`. The `default` password policy, intended for most GeoServer users, does not have any active password constraints. The `master` password policy, intended for the *Root account*, specifies a **minimum password length of eight characters**. Password policies are applied to users via the user/group service.

Clicking an existing policy enables editing, while clicking the *Add new* button will create a new password policy.

13.1.4 Users, Groups, Roles

This section provides the configuration options for *User/group services* and *Role services*. In addition, users, groups, and roles themselves and can be added, edited, or removed. A great deal of configuration can be

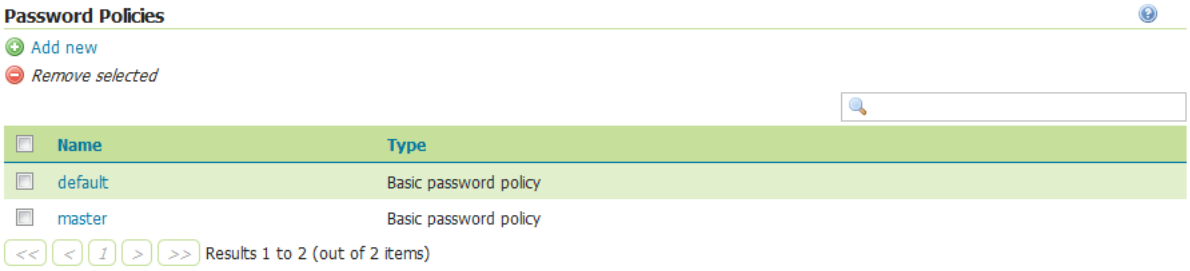


Fig. 13.16: List of password policies

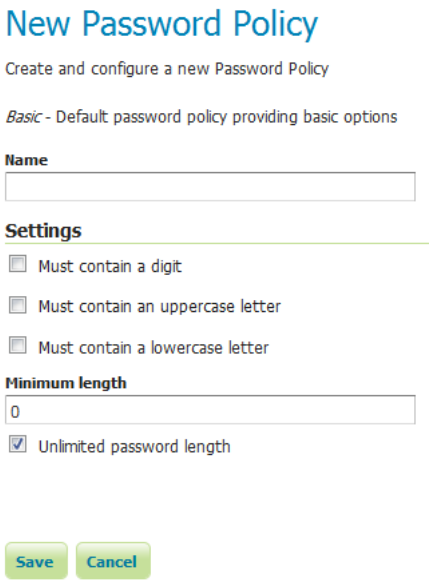


Fig. 13.17: Creating a new password policy

accomplished in this section and related pages.

User Group Services

In this menu, user/group services can be added, removed, or edited. By default, there is one user/group service in GeoServer, which is *XML-based*. It is encrypted with *Weak PBE* and uses the default *password policy*. It is also possible to have a user/group service based on *JDBC*, with or without JNDI.

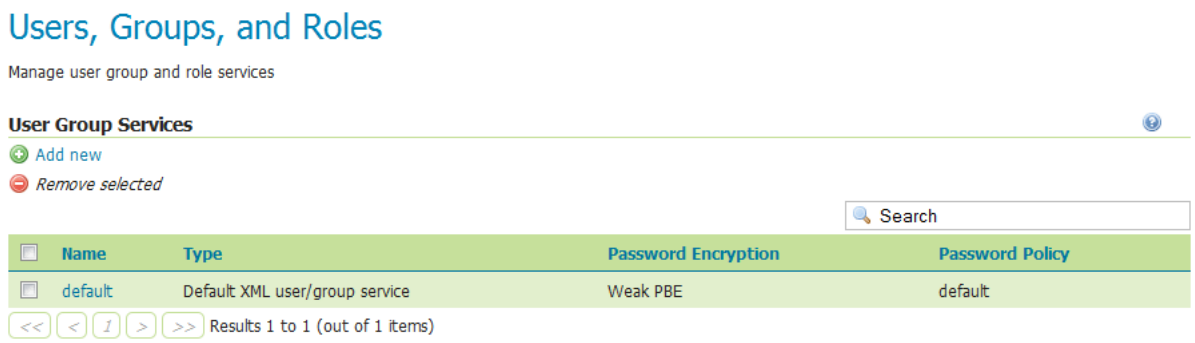


Fig. 13.18: *User/group services*

Clicking an existing user/group service will enable editing, while clicking the *Add new* link will configure a new user/group service.

There are three tabs for configuration: Settings, Users, and Groups.

Note: When creating a new user/group service, the form filled out initially can be found under the Settings tab.

Add new XML user/group service

To add a new XML user/group service, click the *Add new* link. XML is the default option. The following figure shows the configuration options for an XML user/group service.

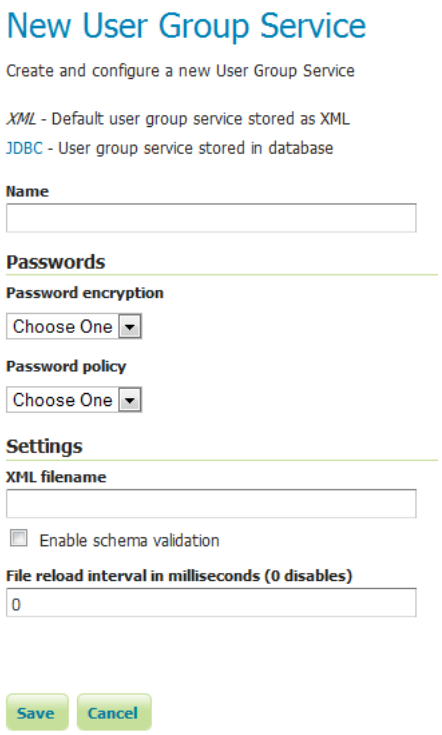


Fig. 13.19: *Adding an XML user/group service*

Option	Description
Name	The name of the user/group service
Password encryption	Sets the type of *Password encryption*. Options are *Plain text*, *Weak PBE*, *Strong PBE*, and *Digest*.
Password policy	Sets the *password policy*. Options are any active password policies as set in the *Passwords* section.
XML filename	Name of the file that will contain the user and group information. Default is `users.xml` in the `security/usergroup/<name_of_usergroupservice>` directory.
Enable schema validation	If selected, forces schema validation to occur every time the XML file is read. This option is useful when editing the XML file by hand.
File reload interval	Defines the frequency (in milliseconds) in which GeoServer will check for changes to the XML file. If the file is found to have been modified, GeoServer will recreate the user/group database based on the current state of the file. This value is meant to be set in cases where the XML file contents might change "out of process" and not directly through the web admin interface. The value is specified in milliseconds. A value of 0 disables any checking of the file.

Add new JDBC user/group service

To add a new XML user/group service, click the *Add new* link, and then the *JDBC* option at the top of the following form. The following figure shows the configuration options for a JDBC user/group service.

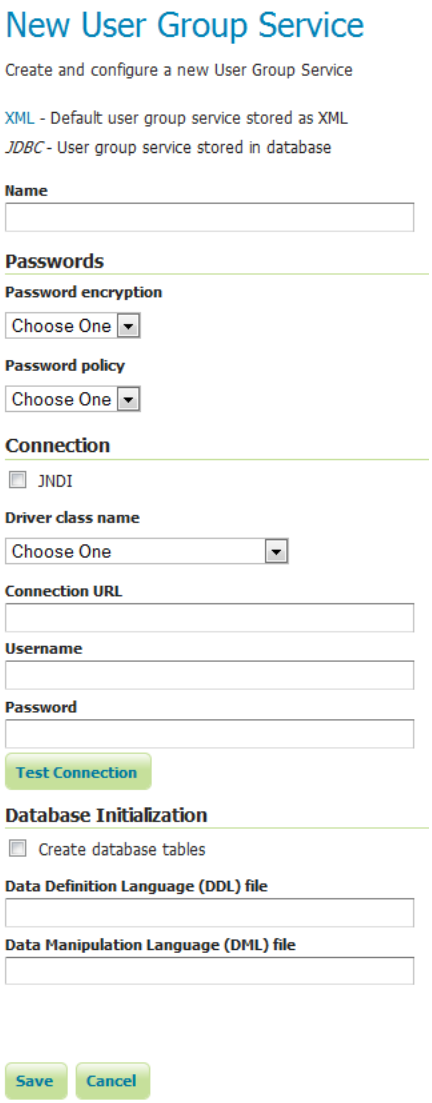


Fig. 13.20: *Adding a user/group service via JDBC*

Option	Description
Name	Name of the JDBC user/group service in GeoServer
Password encryption	The method to used to *encrypt user passwords*
Password policy	The *policy* to use to enforce constraints on user passwords
JNDI	When unchecked, specifies a direct connection to the database. When checked, specifies an existing connection located through *JNDI*.
Driver class name	JDBC driver to use for the database connection
Connection URL	Specifies the JDBC URL to use when creating the database connection
Username	Username to use when connecting to the database
Password	Password to use when connecting to the database
Create database tables	Specifies whether to create all the necessary tables in the underlying database
Data Definition Language (DDL) file	Specifies a custom DDL file to use for creating tables in the underlying database, for cases where the default DDL statements fail on the given database. If left blank, internal defaults are used.
Data Manipulation Language (DML) file	Specifies a custom DML file to use for accessing tables in the underlying database, for cases where the default DML statements fail on the given database. If left blank, internal defaults are used.

In addition to the parameters listed above, the following additional parameter will apply when the *JNDI* flag is set.

Option	Description
JNDI resource name	JNDI name used to locate the database connection.

Edit user/group service

Once the new user/group service is added (either XML or JDBC), clicking on it in the list of user/group services will allow additional options to be specified, such as the users and groups associated with the service.

There are three tabs in the resulting menu: *Settings*, *Users*, and *Groups*. The Settings tab is identical to that found when creating the user/group service, while the others are described below.

The Users tab provides options to configure users in the user/group service.

Clicking a username will allow its parameters to be changed, while clicking the *Add new* link will create a new user.

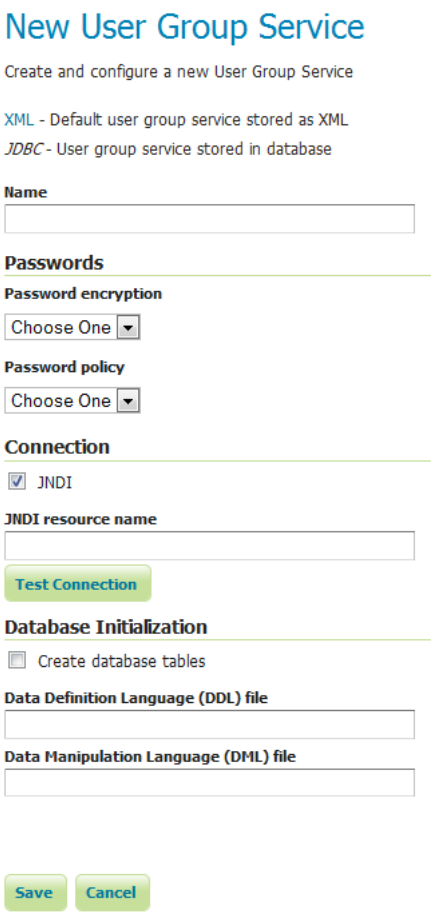


Fig. 13.21: *Adding a user/group service via JDBC with JNDI*

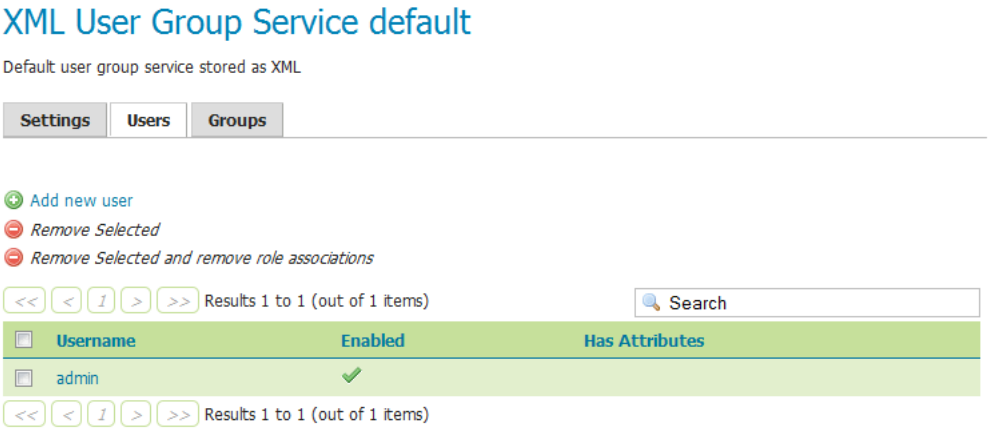


Fig. 13.22: *Users tab*

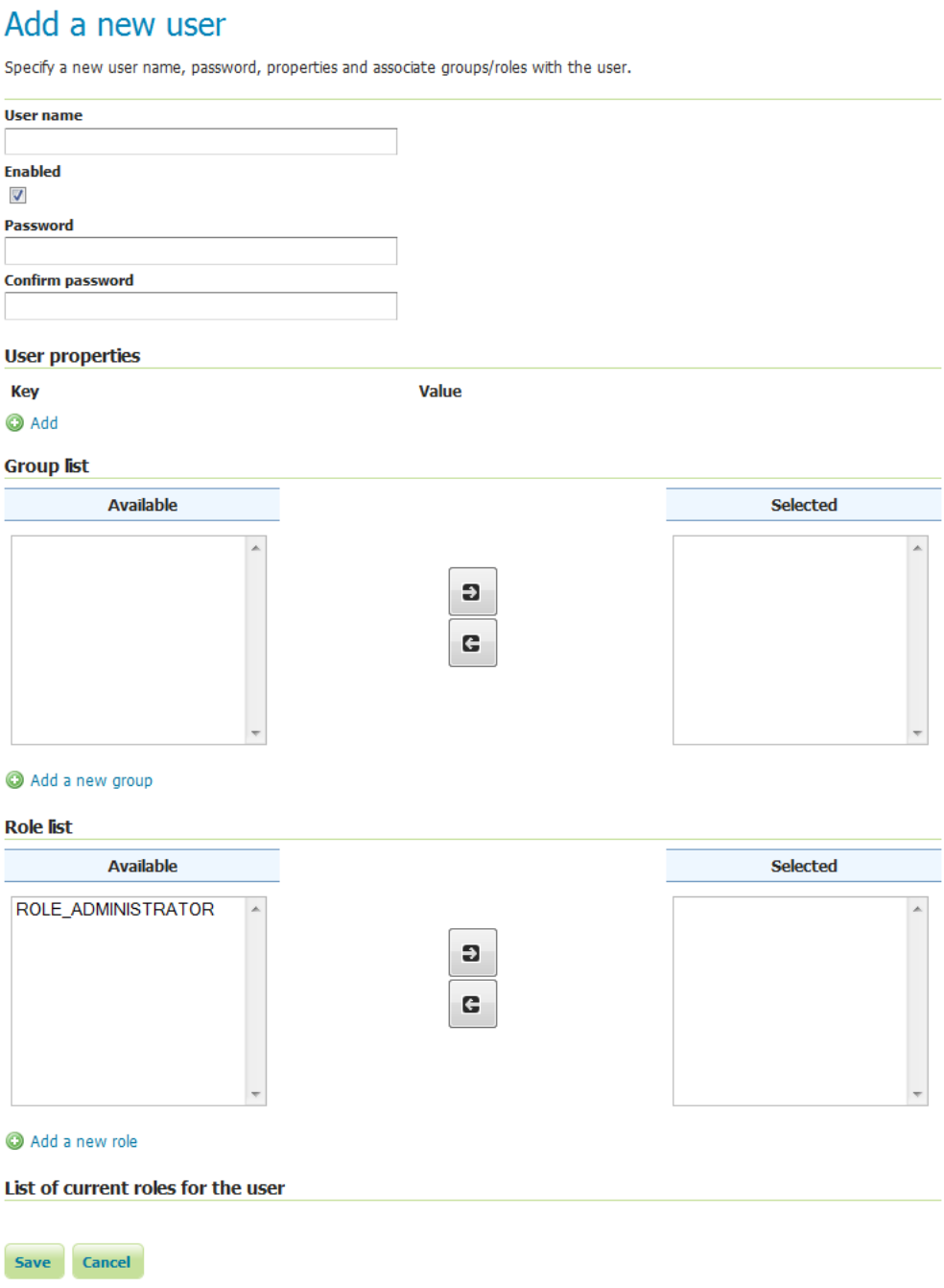


Fig. 13.23: *Creating or editing a user*

Add user

Option	Description
User name	The name of the user
Enabled	When selected, will enable the user to authenticate
Password	The password for this user. Existing passwords will be obscured when viewed.
Confirm password	To set or change the password enter the password twice.
User properties	Key/value pairs associated with the user. Used for associating additional information with the user.
Group list	Full list of groups, including list of groups to which the user is a member. Membership can be toggled here via the arrow buttons.
Add a new group	Shortcut to adding a new group. Also available in the Groups tab.
Role list	Full list of roles, including a list of roles to which the user is associated. Association can be toggled here via the arrow buttons.
Add a new role	Shortcut to adding a new role
List of current roles for the user	List of current roles associated with the user. Click a role to enable editing.

The Groups tab provides configuration options for groups in this user/group service. There are options to add and remove a group, with an additional option to remove a group and the roles associated with that group.

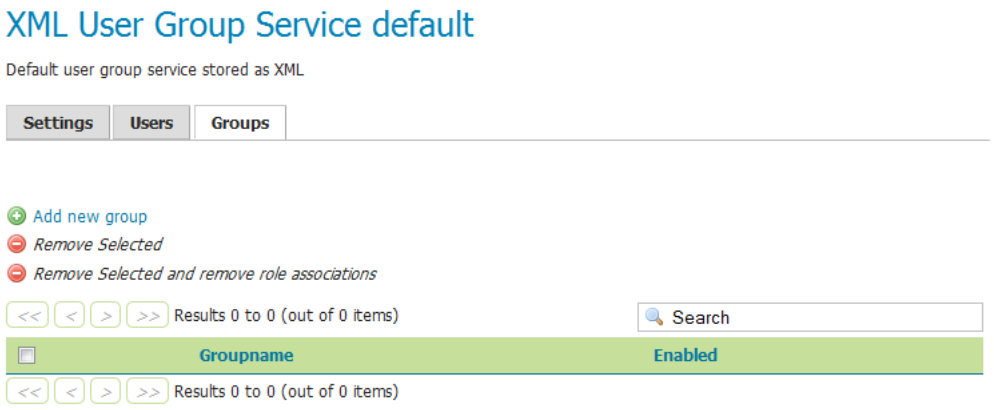


Fig. 13.24: *Groups tab*

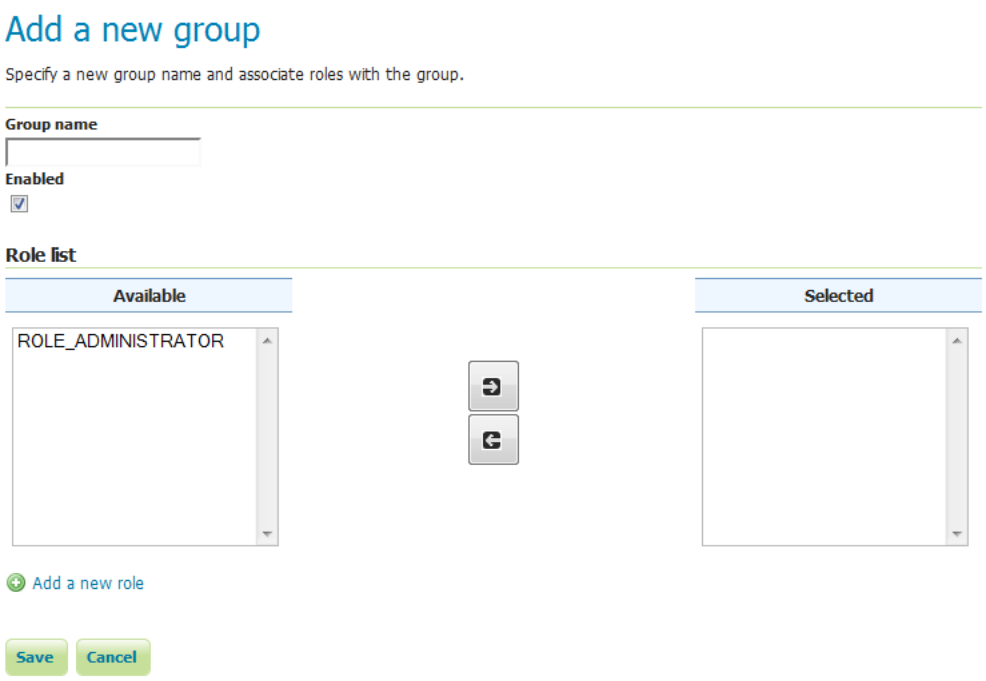


Fig. 13.25: *Creating or editing a group*

Add group

Option	Description
Group name	The name of the group
Enabled	When selected the group will be active
Role list	Full list of roles, including a list of roles to which the group is associated. Association can be toggled here via the arrow buttons.
Add a new role	Shortcut to adding a new role

In this menu, user/group services can be added, removed, or edited. By default, there is one user/group service in GeoServer, which is *XML-based*. It is encrypted with *Weak PBE* and uses the default *password policy*. It is also possible to have a user/group service based on *JDBC* with or without JNDI.

Role services

In this menu, role services can be added, removed, or edited. By default, the active role service in GeoServer is *XML-based*, but it is also possible to have a role service based on *JDBC*, with or without JNDI.

The Administrator role is called `ROLE_ADMINISTRATOR`.

Clicking an existing role service will open it for editing, while clicking the *Add new* link will configure a new role service.

There are two pages for configuration: Settings and Roles.

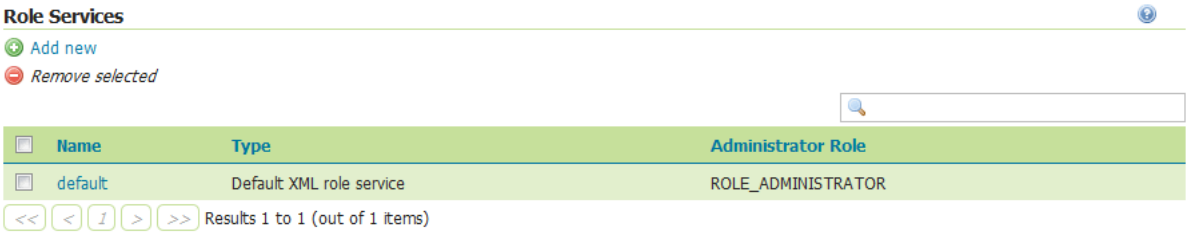


Fig. 13.26: *Role services*

Note: When creating a new role service, the form filled out initially can be found under the Settings tab.

Add new XML role service

To add a new XML role service, click the *Add new* link. XML is the default option. The following figure shows the configuration options for an XML role service.

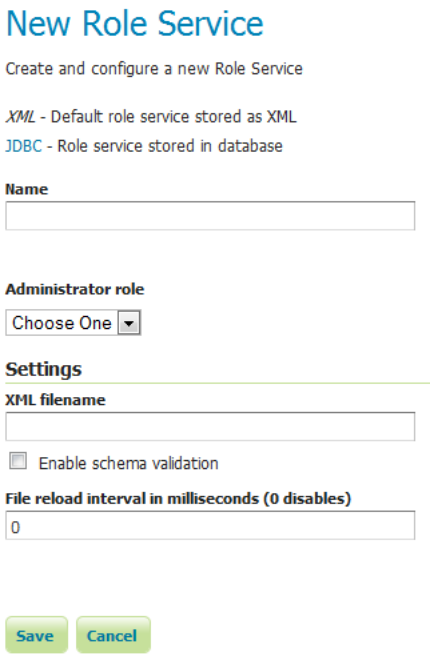


Fig. 13.27: *Adding an XML role service*

Option	Description
Name	The name of the role service
Administrator role	The name of the role that performs the administrator functions
XML filename	Name of the file that will contain the role information. Default is `roles.xml` in the `security/role/<name_of_roleservice>` directory.
File reload interval	Defines the frequency (in milliseconds) in which GeoServer will check for changes to the XML file. If the file is found to have been modified, GeoServer will recreate the user/group database based on the current state of the file. This value is meant to be set in cases where the XML file contents might change "out of process" and not directly through the web admin interface. The value is specified in milliseconds. A value of 0 disables any checking of the file.

Add new JDBC role service

To add a new XML role service, click the *Add new* link, and then the *JDBC* option at the top of the following form. The following figure shows the configuration options for a JDBC role service.

Option	Description
Name	Name of the JDBC role service in GeoServer
Administrator role	The name of the role that performs the administrator function
JNDI	When unchecked, specifies a direct connection to the database. When checked, specifies an existing connection located through *JNDI*.
Driver class name	JDBC driver to use for the database connection
Connection URL	Specifies the JDBC URL to use when creating the database connection
Username	Username to use when connecting to the database
Password	Password to use when connecting to the database
Create database tables	Specifies whether to create all the necessary tables in the underlying database
Data Definition Language (DDL) file	Specifies a custom DDL file to use for creating tables in the underlying database, for cases where the default DDL statements fail on the given database. If left blank, internal defaults are used.
Data Manipulation Language (DML) file	Specifies a custom DML file to use for accessing tables in the underlying database, for cases where the default DML statements fail on the given database. If left blank, internal defaults are used.

In addition to the parameters listed above, the following additional parameter will apply when the *JNDI* flag is set.

Option	Description
JNDI resource name	JNDI name used to locate the database connection.

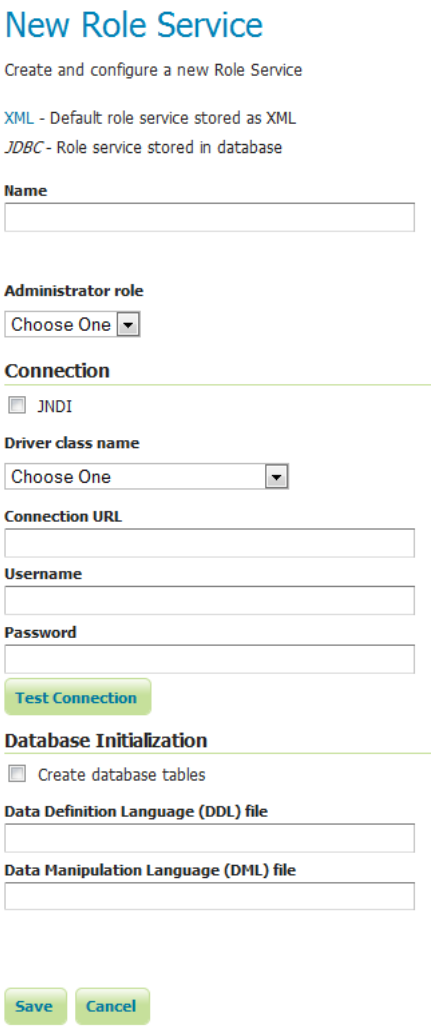


Fig. 13.28: *Adding a role service via JDBC*

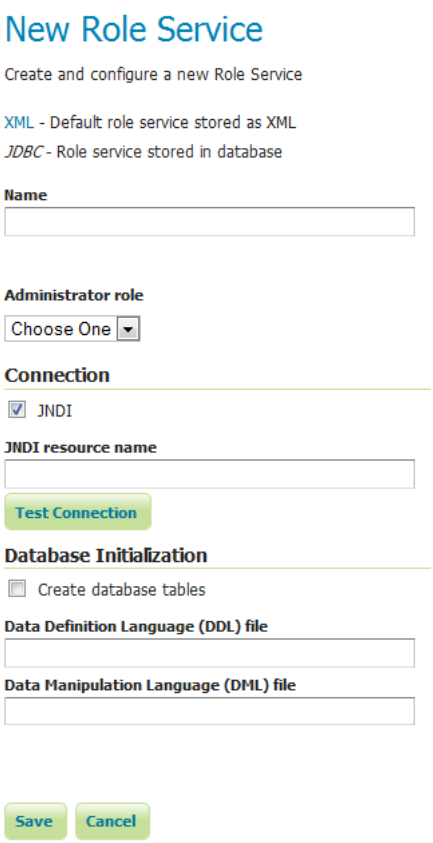


Fig. 13.29: *Adding a role service via JDBC with JNDI*

Add new LDAP role service

To add a new LDAP role service, click the *Add new* link, and then the *LDAP* option at the top of the following form. The following figure shows the configuration options for a LDAP role service.

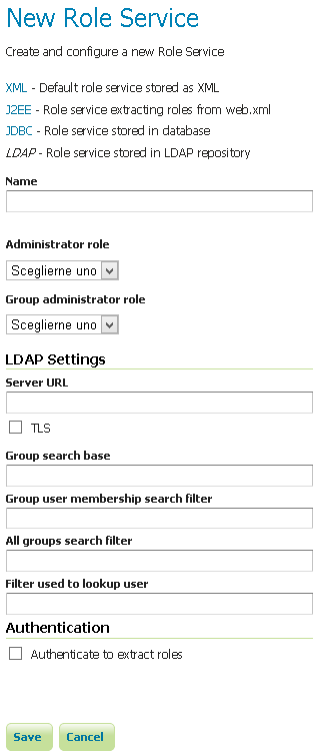


Fig. 13.30: *Adding a role service via LDAP*

Option	Description
Name	Name of the LDAP role service in GeoServer
Administrator role	The name of the role that performs the administrator function
Group administrator role	The name of the role that performs the group administrator function
Server URL	URL for the LDAP server connection. It must include the protocol, host, and port, as well as the "distinguished name" (DN) for the root of the LDAP tree.
TLS	Enables a STARTTLS connection. (See the section on *Secure LDAP connections*.)
Group search base	Relative name of the node in the tree to use as the base for LDAP groups. Example: `ou=groups`. The root DN specified as port of the *Server URL* is automatically appended.
Group user membership search filter	Search pattern for extracting users of a LDAP group a user belongs to. This may contain some placeholder values: `{0}`, the `username` of the user, for example `bob`. `{1}`, the full DN of the user, for example `uid=bob,ou=users`. To use this placeholder, the *Filter used to lookup user* needs to be defined, so that the dn of a user can be extracted from its username.
All groups search filter	Search pattern for locating the LDAP groups to be mapped to GeoServer roles inside the *Group search base* root node
Filter used to lookup user.	optional filter used to extract a user dn, to be used together with *Group user membership search filter* when the {1} placeholder is specified. This may contain a placeholder value: `{0}`, the `username` of the user, for example `bob`.
Authenticate to extract roles	When checked all LDAP searches will be done in authenticated mode, using the credentials given with the *Username* and *Password* options
Username	Username to use when connecting to the LDAP server. Only applicable when the *Authenticate to extract roles* parameter is **checked**.
Password	Password to use when connecting to the LDAP server. Only applicable when the *Authenticate to extract roles* parameter is **checked**.

Edit role service

Once the new role service is added (either XML or JDBC), clicking it in the list of role services will allow the additional options to be specified, such as the roles associated with the service.

There are two tabs in the resulting menu: *Settings* and *Roles*. The Settings tab is identical to that found when creating the role service, while the Roles tab is described below.

Clicking a role will allow its parameters to be changed, while clicking the *Add new* link will create a new role.

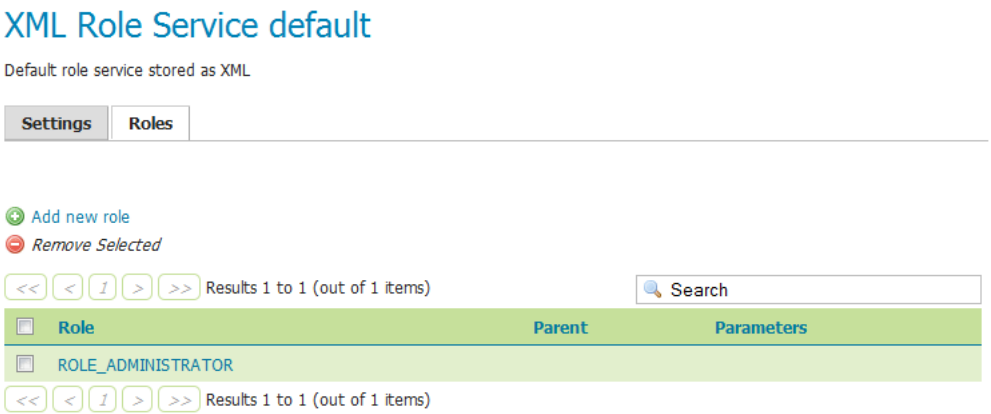


Fig. 13.31: *Roles tab*

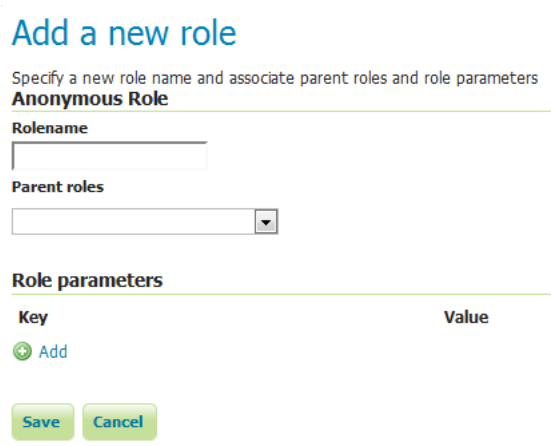


Fig. 13.32: *Creating or editing a role*

Add role

Option	Description
Role name	The name of role. Convention is uppercase, but is not required.
Parent roles	The role that this role inherits. See the section on *Roles* for more information on inheritance.
Role parameters	Key/value pairs associated with the role. Used for associating additional information with the role.

13.1.5 Data

This section provides access to security settings related to data management and *Layer security*. Data access is granted to roles, and roles are granted to users and groups.

Rules

There are two rules available by default, but they don't provide any restrictions on access by default. The first rule `*.*.r`, applied to all roles, states that any operation in any resource in any workspace can be read. The second rule, `*.*.w`, also applied to all roles, says the same for write access.

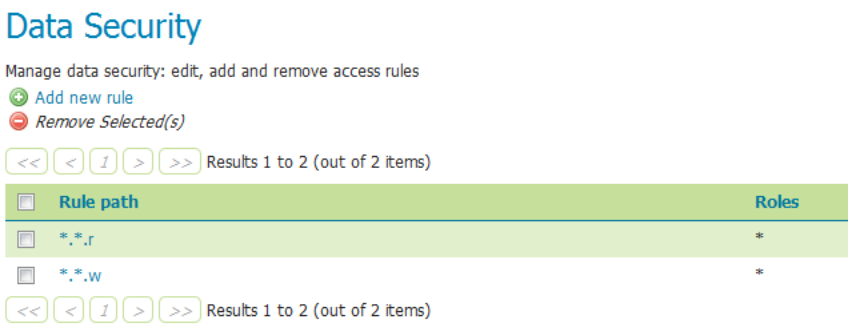


Fig. 13.33: *Rules for data access*

Clicking an existing rule will open it for editing, while clicking the *Add a new rule* link will create a new rule.

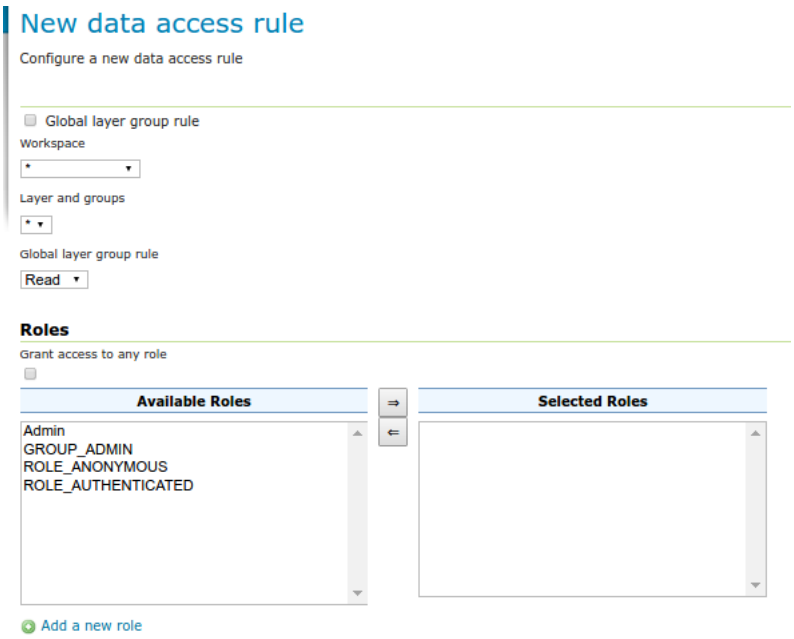


Fig. 13.34: *Creating a new rule*

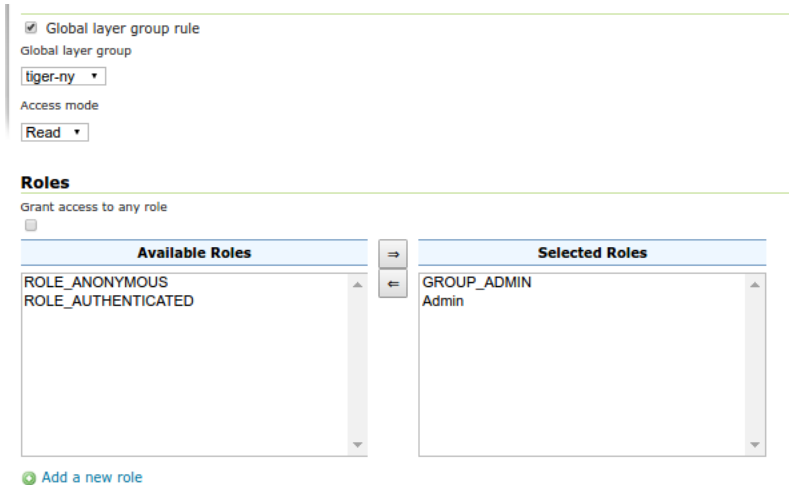


Fig. 13.35: *Editing a layer group rule*

Option	Description
Global layer group rule	If checked, switches the editor to create/edit a rule about a global layer group (and will remove the layer configuration as a result)
Workspace	Sets the allowed workspace for this rule. Options are * (all workspaces), or the name of each workspace.
Layer and groups	Sets the allowed layer/groups for this rule. Options are * (all layers/groups in the chosen workspace), or the name of each layer in the above workspace. Will be disabled until the workspace is set.
Access mode	Specifies whether the rule refers to either `Read` or `Write` mode
Grant access to any role	If selected, the rule will apply to all roles, with no need to specify
Role list	Full list of roles, including a list of roles to which the rule is associated. Association can be toggled here via the arrow buttons. This option is not applied if *Grant access to any role* is checked.
Add a new role	Shortcut to adding a new role

Catalog Mode

This mode configures how GeoServer will advertise secured layers and behave when a secured layer is accessed without the necessary privileges. There are three options: *HIDE*, *MIXED*, and *CHALLENGE*. For further information on these options, please see the section on *Layer security*.

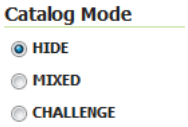


Fig. 13.36: *Catalog mode*

13.1.6 Services

This section provides access to the settings for *Service Security*. GeoServer can limit access based on OWS services (WFS, WMS, etc.) and their specific operations (GetCapabilities, GetMap, and so on).

By default, no service-based security is in effect in GeoServer. However rules can be added, removed, or edited here.

Clicking the *Add a new rule* link will create a new rule.

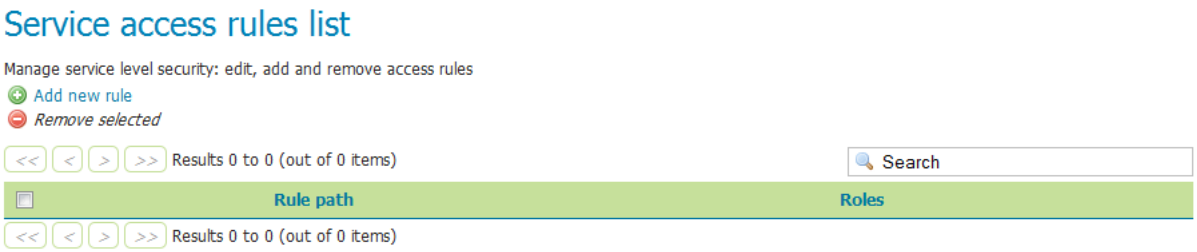


Fig. 13.37: Service access rules list

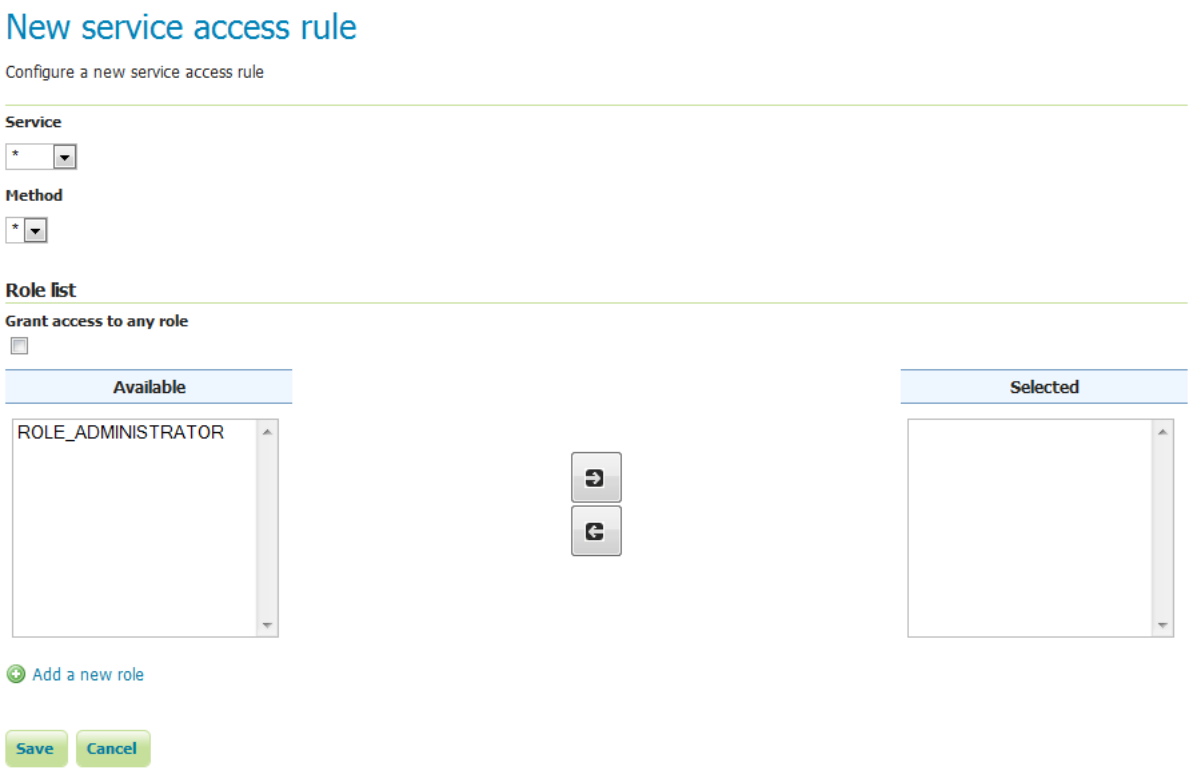


Fig. 13.38: New service rule

Option	Description
Service	Sets the OWS service for this rule. Options are `*`, meaning all services, `wcs`, `wfs`, or `wms`.
Method	Sets the specific operation for this rule. Options depend on the *Service*, but include `*`, meaning all operations, as well as every service operation known to GeoServer, such as Capabilities, Transaction, GetMap, and more.
Grant access to any role	If selected, the rule will apply to all roles (no need to specify which ones)
Role list	Full list of roles, including a list of roles to which the rule is associated. Association can be switched here via the arrow buttons. This option is not applied if *Grant access to any role* is checked.
Add a new role	Shortcut to adding a new role

13.1.7 File Browsing

The GeoServer web admin employs a file browser dialog that will expose locations of the file system other than the GeoServer directory. These locations include the root of the file system and the users home directory. In highly secure and multi-tenant environments disabling this feature may be desired.

The property `GEOSERVER_FILEBROWSER_HIDEFS` can be used to disable this functionality. When set to `true` only the GeoServer data directory will be exposed through the file browser.

The property is set through one of the standard means:

- `web.xml`

```
<context-param>
  <param-name>GEOSERVER_FILEBROWSER_HIDEFS</param-name>
  <param-value>true</param-value>
</context-param>
```

- System property

```
-DGEOSERVER_FILEBROWSER_HIDEFS=true
```

- Environment variable

```
export GEOSERVER_FILEBROWSER_HIDEFS=true
```

13.2 Role system

Security in GeoServer is based on a **role-based system**, with roles created to serve particular functions. Examples of roles sporting a particular function are those accessing the Web Feature Service (WFS), administering the *Web administration interface*, and reading a specific layer. Roles are assigned to users and groups of users, and determine what actions those users or groups are permitted to do. A user is authorized through *Authentication*.

13.2.1 Users and Groups

The definition of a GeoServer **user** is similar to most security systems. Although the correct Java term is **principal**—a principal being a human being, computer, software system, and so on—the term **user** is adopted throughout the GeoServer documentation. For each user the following information is maintained:

- User name
- *Password* (optionally stored *encrypted*)
- A flag indicating if the user is enabled (this is the default). A disabled user is prevented from logging on. Existing user sessions are not affected.
- Set of key/value pairs

Key/value pairs are implementation-specific and may be configured by the *user/group service* the user or group belongs to. For example, a user/group service that maintains information about a user such as Name, Email address, and so on, may wish to associate those attributes with the user object.

A GeoServer **group** is simply a set of users. For each group the following information is maintained:

- Group name
- A flag indicating if the group is enabled (this is the default). A disabled group does not contribute to the role calculation for all users contained in this group.
- List of users who belong to the group

13.2.2 User/group services

A **user/group service** provides the following information for users and groups:

- Listing of users
- Listing of groups, including users affiliated with each group
- User passwords

Many authentication providers will make use of a user/group service to perform authentication. In this case, the user/group service would be the database against which users and passwords are authenticated. Depending on how the *Authentication chain* is configured, there may be zero, one, or multiple user/group services active at any given time.

A user/group service may be read-only, providing access to user information but not allowing new users and groups to be added or altered. This may occur if a user/group service was configured to delegate to an external service for the users and groups database. An example of this would be an external LDAP server.

By default, GeoServer support two types of user/group services:

- XML—*(Default)* User/group service persisted as XML
- JDBC—User/group service persisted in database via JDBC

XML user/group service

The XML user/group service persists the user/group database in an XML file. This is the default behavior in GeoServer. This service represents the user database as XML, and corresponds to this `XML schema`.

Note: The XML user/group file, `users.xml`, is located in the GeoServer data directory, `security/usergroup/<name>/users.xml`, where `<name>` is the name of the user/group service.

The following is the contents of `users.xml` that ships with the default GeoServer configuration:

```
<userRegistry version="1.0" xmlns="http://www.geoserver.org/security/users">
  <users>
    <user enabled="true" name="admin" password=
↪"crypt1:5WK8hBrtrte9wtImg5i5fjnd8VeqCjDB"/>
  </users>
  <groups/>
</userRegistry>
```

This particular configuration defines a single user, `admin`, and no groups. By default, stored user passwords are encrypted using the *weak PBE* method.

For further information, please refer to *configuring a user/group service* in the *Web administration interface*.

JDBC user/group service

The JDBC user/group service persists the user/group database via JDBC, managing the user information in multiple tables. The user/group database schema is as follows:

Table 13.1: Table: users

Field	Type	Null	Key
name	varchar(128)	NO	PRI
password	varchar(254)	YES	
enabled	char(1)	NO	

Table 13.2: Table: user_props

Field	Type	Null	Key
username	varchar(128)	NO	PRI
propname	varchar(64)	NO	PRI
propvalue	varchar(2048)	YES	

Table 13.3: Table: groups

Field	Type	Null	Key
name	varchar(128)	NO	PRI
enabled	char(1)	NO	

Table 13.4: Table: group_members

Field	Type	Null	Key
groupname	varchar(128)	NO	PRI
username	varchar(128)	NO	PRI

The `users` table is the primary table and contains the list of users with associated passwords. The `user_props` table maps additional properties to a user. (See *Users and Groups* for more details.) The `groups` table lists all available groups, and the `group_members` table maps which users belong to which groups.

The default GeoServer security configuration is:

Table 13.5: Table: users

name	password	enabled
Empty	*Empty*	*Empty*

Table 13.6: Table: user_props

username	propname	propvalue
Empty	*Empty*	*Empty*

Table 13.7: Table: groups

name	enabled
Empty	*Empty*

Table 13.8: Table: group_members

groupname	username
Empty	*Empty*

For further information, please refer to *configuring a user/group service* in the *Web administration interface*.

LDAP user/group service

The LDAP user/group service is a read only user/group service that maps users and groups from an LDAP repository to GeoServer users and groups.

Users are extracted from a specific LDAP node, configured as the `Users search base`. Groups are extracted from a specific LDAP node, configured as the `Groups search base`. A user is mapped for every matching user and a group is mapped for every matching group.

It is possible to specify the attributes which contain the name of the group (such as `cn`), the user (such as `uid`) as well as the membership relationship between the two (such as `member`). However, it is also possible to specify specific filters to search for all users/groups (for example `cn=*`), find a user/group by name (for example `cn={0}`) and map users to groups (such as `member={0}`). These filters can also be automatically derived from the attribute names. Alternatively, the attribute names may be left empty if the filters are provided.

For users, additional properties (key/value pairs, see *Users and Groups*) may be populated from the LDAP Server by providing a comma separated list of property names.

Retrieving the user/group information can be done anonymously or using a given username/password if the LDAP repository requires it.

An example of configuration file (config.xml) for this type of role service is the following:

```
<org.geoserver.security.ldap.LDAPUserGroupServiceConfig>
  <id>2c3e0e8d:154853796a3:-8000</id>
  <name>myldapservice</name>
  <className>org.geoserver.security.ldap.LDAPUserGroupService</className>
  <serverURL>ldap://127.0.0.1:10389/dc=acme,dc=org</serverURL>
  <groupSearchBase>ou=groups</groupSearchBase>
  <groupFilter>cn={0}</groupFilter>
  <groupNameAttribute>cn</groupNameAttribute>
```

```
  <allGroupsSearchFilter>cn=*</allGroupsSearchFilter>
  <groupSearchFilter>member={0}</groupSearchFilter>
  <groupMembershipAttribute>member</groupMembershipAttribute>
  <userSearchBase>ou=people</userSearchBase>
  <userFilter>uid</userFilter>
  <userNameAttribute>uid={0}</userNameAttribute>
  <allUsersSearchFilter>uid=*</allUsersSearchFilter>
  <useTLS>false</useTLS>
  <bindBeforeGroupSearch>true</bindBeforeGroupSearch>
  <user>admin</user>
  <password>admin</password>
  <passwordEncoderName>emptyPasswordEncoder</passwordEncoderName>
  <passwordPolicyName>default</passwordPolicyName>
  <populatedAttributes>email, telephone</populatedAttributes>
</org.geoserver.security.ldap.LDAPUserGroupServiceConfig>
```

For further information, please refer to *configuring a user/group service* in the *Web administration interface*.

13.2.3 Roles

GeoServer **roles** are keys associated with performing certain tasks or accessing particular resources. Roles are assigned to users and groups, authorizing them to perform the actions associated with the role. A GeoServer role includes the following:

- Role name
- Parent role
- Set of key/value pairs

GeoServer roles support inheritance—a child role inherits all the access granted to the parent role. For example, suppose you have one role named `ROLE_SECRET` and another role, `ROLE_VERY_SECRET`, that extends `ROLE_SECRET`. `ROLE_VERY_SECRET` can access everything `ROLE_SECRET` can access, but not vice versa.

Key/value pairs are implementation-specific and may be configured by the *role service* the user or group belongs to. For example, a role service that assigns roles based on employee organization may wish to associate additional information with the role such as Department Name.

GeoServer has a number of system roles, the names of which are reserved. Adding a new GeoServer role with reserved name is not permitted.

- `ROLE_ADMINISTRATOR`—Provides access to all operations and resources
- `ROLE_GROUP_ADMIN`—Special role for administrating user groups
- `ROLE_AUTHENTICATED`—Assigned to every user authenticating successfully
- `ROLE_ANONYMOUS`—Assigned if anonymous authentication is enabled and user does not log on

13.2.4 Role services

A **role service** provides the following information for roles:

- List of roles
- Calculation of role assignments for a given user
- Mapping of a role to the system role `ROLE_ADMINISTRATOR`

- Mapping of a role to the system role `ROLE_GROUP_ADMIN`

When a user/group service loads information about a user or a group, it delegates to the role service to determine which roles should be assigned to the user or group. Unlike *User/group services*, only one role service is active at any given time.

By default, GeoServer supports two types of role services:

- XML—*(Default)* role service persisted as XML
- JDBC—Role service persisted in a database via JDBC

Mapping roles to system roles

To assign the system role `ROLE_ADMINISTRATOR` to a user or to a group, a new role with a different name must be created and mapped to the `ROLE_ADMINISTRATOR` role. The same holds true for the system role `ROLE_GROUP_ADMIN`. The mapping is stored in the service's `config.xml` file.

```
<roleService>
  <id>471ed59f:13915c479bc:-7ffc</id>
  <name>default</name>
  <className>org.geoserver.security.xml.XMLRoleService</className>
  <fileName>roles.xml</fileName>
  <checkInterval>10000</checkInterval>
  <validating>true</validating>
  <adminRoleName>ADMIN</adminRoleName>
  <groupAdminRoleName>GROUP_ADMIN</groupAdminRoleName>
</roleService>
```

In this example, a user or a group assigned to the role `ADMIN` is also assigned to the system role `ROLE_ADMINISTRATOR`. The same holds true for `GROUP_ADMIN` and `ROLE_GROUP_ADMIN`.

XML role service

The XML role service persists the role database in an XML file. This is the default role service for GeoServer. This service represents the user database as XML, and corresponds to this `XML schema`.

Note: The XML role file, `roles.xml`, is located in the GeoServer data directory, `security/role/<name>/roles.xml`, where `<name>` is the name of the role service.

The service is configured to map the local role `ADMIN` to the system role `ROLE_ADMINISTRATOR`. Additionally, `GROUP_ADMIN` is mapped to `ROLE_GROUP_ADMIN`. The mapping is stored the `config.xml` file of each role service.

The following provides an illustration of the `roles.xml` that ships with the default GeoServer configuration:

```
<roleRegistry version="1.0" xmlns="http://www.geoserver.org/security/roles">
  <roleList>
    <role id="ADMIN"/>
    <role id="GROUP_ADMIN"/>
  </roleList>
  <userList>
    <userRoles username="admin">
      <roleRef roleID="ADMIN"/>
    </userRoles>
```

```
  </userList>
  <groupList/>
</roleRegistry>
```

This configuration contains two roles named `ADMIN` and `GROUP_ADMIN`. The role `ADMIN` is assigned to the `admin` user. Since the `ADMIN` role is mapped to the system role `ROLE_ADMINISTRATOR`, the role calculation assigns both roles to the `admin` user.

For further information, please refer to *configuring a role service* in the *Web administration interface*.

J2EE role service

The J2EE role service parses roles from the `WEB-INF/web.xml` file. As a consequence, this service is a read only role service. Roles are extracted from the following XML elements:

`<security-role>`

```
<security-role>
   <role-name>role1</role-name>
</security-role>
<security-role>
   <role-name>role2</role-name>
</security-role>
<security-role>
   <role-name>employee</role-name>
</security-role>
```

Roles retrieved:

- `role1`
- `role2`
- `employee`

`<security-constraint>`

```
 <security-constraint>
    <web-resource-collection>
        <web-resource-name>Protected Area</web-resource-name>
        <url-pattern>/jsp/security/protected/*</url-pattern>
        <http-method>PUT</http-method>
        <http-method>DELETE</http-method>
        <http-method>GET</http-method>
        <http-method>POST</http-method>
    </web-resource-collection>
    <auth-constraint>
        <role-name>role1</role-name>
        <role-name>employee</role-name>
    </auth-constraint>
</security-constraint>
```

Roles retrieved:

- `role1`

- employee

<security-role-ref>

```
<security-role-ref>
    <role-name>MGR</role-name>
    <!-- role name used in code -->
    <role-link>employee</role-link>
  </security-role-ref>
```

Roles retrieved:

- MGR

JDBC role service

The JDBC role service persists the role database via JDBC, managing the role information in multiple tables. The role database schema is as follows:

Table 13.9: Table: roles

Field	Type	Null	Key
name	varchar(64)	NO	PRI
parent	varchar(64)	YES	

Table 13.10: Table: role_props

Field	Type	Null	Key
rolename	varchar(64)	NO	PRI
propname	varchar(64)	NO	PRI
propvalue	varchar(2048)	YES	

Table 13.11: Table: user_roles

Field	Type	Null	Key
username	varchar(128)	NO	PRI
rolename	varchar(64)	NO	PRI

Table 13.12: Table: group_roles

Field	Type	Null	Key
groupname	varchar(128)	NO	PRI
rolename	varchar(64)	NO	PRI

The `roles` table is the primary table and contains the list of roles. Roles in GeoServer support inheritance, so a role may optionally have a link to a parent role. The `role_props` table maps additional properties to a role. (See the section on *Roles* for more details.) The `user_roles` table maps users to the roles they are assigned. Similarly the `group_roles` table maps which groups have been assigned to which roles.

The default GeoServer security configuration is:

Table 13.13: Table: roles

name	parent
Empty	*Empty*

Table 13.14: Table: role_props

rolename	propname	propvalue
Empty	*Empty*	*Empty*

Table 13.15: Table: user_roles

username	rolename
Empty	*Empty*

Table 13.16: Table: group_roles

groupname	rolename
Empty	*Empty*

For further information, please refer to *configuring a role service* in the *Web administration interface*.

LDAP role service

The LDAP role service is a read only role service that maps groups from an LDAP repository to GeoServer roles.

Groups are extracted from a specific LDAP node, configured as the `Groups search base`. A role is mapped for every matching group. The role will have a name that is built taking the Group common name (cn attribute), transformed to upper case and with a `ROLE_` prefix applied.

It is possible to filter extracted groups using an `All groups filter` (defaults to `cn=*` that basically extracts all nodes from the configured base). It is also possible to configure the filter for `users to roles membership` (defaults to member={0}).

A specific group can be assigned to the `ROLE_ADMINISTRATOR` and/or the `ROLE_GROUP_ADMIN` administrative roles.

Groups extraction can be done anonymously or using a given username/password if the LDAP repository requires it.

An example of configuration file (config.xml) for this type of role service is the following:

```
<org.geoserver.security.ldap.LDAPRoleServiceConfig>
  <id>-36dfbd50:1424687f3e0:-8000</id>
  <name>ldapacme</name>
  <className>org.geoserver.security.ldap.LDAPRoleService</className>
  <serverURL>ldap://127.0.0.1:10389/dc=acme,dc=org</serverURL>
  <groupSearchBase>ou=groups</groupSearchBase>
  <groupSearchFilter>member=uid={0},ou=people,dc=acme,dc=org</
↪groupSearchFilter>
  <useTLS>false</useTLS>
  <bindBeforeGroupSearch>true</bindBeforeGroupSearch>
  <adminGroup>ROLE_ADMIN</adminGroup>
  <groupAdminGroup>ROLE_ADMIN</groupAdminGroup>
  <user>uid=bill,ou=people,dc=acme,dc=org</user>
```

```
  <password>hello</password>
  <allGroupsSearchFilter>cn=*</allGroupsSearchFilter>
</org.geoserver.security.ldap.LDAPRoleServiceConfig>
```

For further information, please refer to *configuring a role service* in the *Web administration interface*.

REST role service

The REST role service is a read only role service that maps groups and associated users to roles from a remote REST web service.

The REST service **must** support JSON encoding.

Here is a listing of significant methods provided by the REST Role Service (based on the LDAP role service, which similarly has to make network calls to work):

Table 13.17: Table: roles

Method	Mandatory
getUserNamesForRole(roleName)	N (implemented in LDAP, but I don't see actual users of this method besides a utility method that nobody uses)
getRolesForUser(user)	Y
getRolesForGroup(group)	N
getRoles()	Y (used by the UI)
getParentRole(role)	N
getAdminRole()	Y
getGroupAdminRole()	Y
getRoleCount()	Y (does not seem to be used much, we can trivially implement it from getRoles()

REST APIs

The following is an example of the REST API the role service may handle. The JSON and remote endpoints may differ; this is conifgurable via UI, allowing the REST role service to connect to a generci REST Service

From the above we could have the following REST API to talk to

`../api/roles`

Returns the full list of roles (no paging required, we assume it's small). Example response:

```
{"groups":["r1","r2","r3"]}
```

`../api/adminrole`

Returns the role of the administrator (yes, just one, it's strange...):

```
{"adminRole":["root"]}
```

`../api/users/<user>`

Returns the list of roles for a particular user. Example response:

```
{"users": [{"user":"u1", "groups":["r1","r2"]}]}
```

Configurable API

The GeoServerRoleService talking to a remote service provides the following config parameters:

- Base URL for the remote service
- Configurable URLs for the various calls
- JSON paths to the properties that contain the list of roles, and the one admin role

The above can be configured via the *Web administration interface*. The figure below shows the REST role service options configured to be compatible with the sampe APIs above:

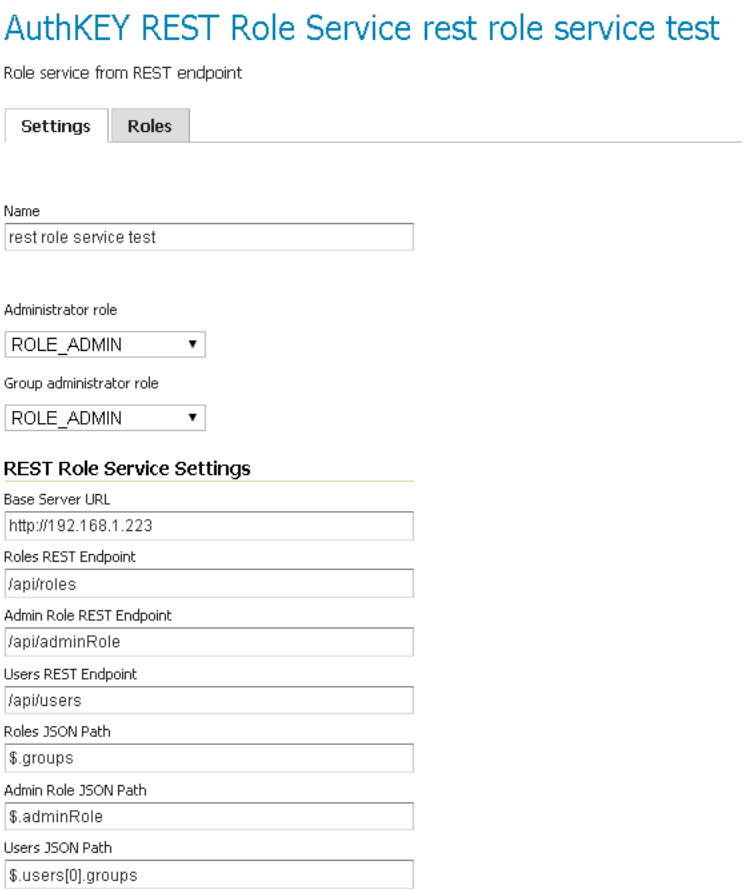


Fig. 13.39: *REST based role service configuration panel*

13.2.5 Role source and role calculation

Different authentication mechanisms provide different possibilities where to look for the roles of a principal/user. The role source is the base for the calculation of the roles assigned to the authenticated principal.

Using a user/group Service

During configuration of an authentication mechanism, the name of a user group service has to be specified. The used role service is always the role service configured as active role service. The role calculation itself

is described here *Interaction between user/group and role services*

Using a role service directly

During configuration of an authentication mechanism, the name of a role service has to be specified. The calculation of the roles works as follows:

1. Fetch all roles for the user.
2. For each role in the result set, fetch all ancestor roles and add those roles to the result set.
3. If the result set contains the local admin role, add the role `ROLE_ADMINISTRATOR`.
4. If the result set contains the local group admin role, add the role `ROLE_GROUP_ADMIN`.

This algorithm does not offer the possibility to have personalized roles and it does not consider group memberships.

Using an HTTP header attribute

The roles for a principal are sent by the client in an HTTP header attribute (Proxy authentication). GeoServer itself does no role calculation and extracts the roles from the header attribute. During configuration, the name of the header attribute must be specified. An example with a header attribute named "roles":

```
roles: role_a;role_b;role_c
```

An example for roles with role parameters:

```
roles: role_a;role_b(pnr=123,nick=max);role_c
```

The default syntax is

- roles are delimited by **;**
- a role parameter list starts with **(** and ends with **)**
- a role parameter is a key value pair delimited by **=**
- role parameters are delimited by **,**

13.2.6 Interaction between user/group and role services

The following section describes the interaction between the *User/group services* and the *Role services*.

Calculating the roles of a user

The diagram below illustrates how a user/group service and a role service interact to calculate user roles.

On fetching an enabled user from a user/group service, the roles(s) assigned to that user must be identified. The identification procedure is:

1. Fetch all enabled groups for the user. If a group is disabled, it is discarded.
2. Fetch all roles associated with the user and add the roles to the result set.
3. For each enabled group the user is a member of, fetch all roles associated with the group and add the roles to the result set.
4. For each role in the result set, fetch all ancestor roles and add those roles to the result set.

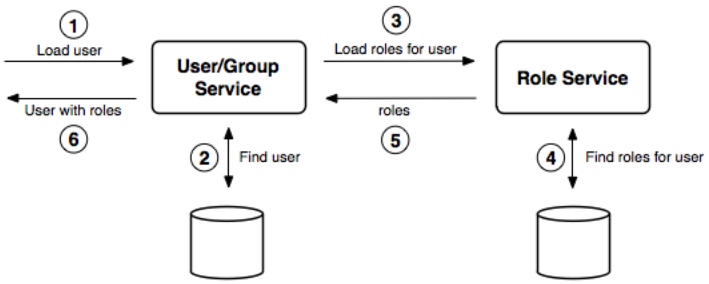


Fig. 13.40: *User/group and role service interacting for role calculation*

5. Personalize each role in the result set as required.
6. If the result set contains the local admin role, add the role `ROLE_ADMINISTRATOR`.
7. If the result set contains the local group admin role, add the role `ROLE_GROUP_ADMIN`.

Note: Role personalization looks for role parameters (key/value pairs) for each role and checks if the user properties (key/value pairs) contain an identical key. If any matches are found, the value of the role parameter is replaced by the value of the user property.

Authentication of user credentials

A user/group service is primarily used during authentication. An authentication provider in the *Authentication chain* may use a user/group service to authenticate user credentials.

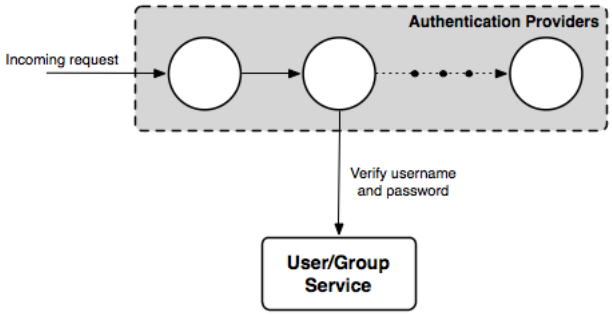


Fig. 13.41: *Using a a user/group service for authentication*

GeoServer defaults

The following diagram illustrates the default user/group service, role service, and authentication provider in GeoServer:

Two authentication providers are configured—the *Root* provider and the *Username/password* provider. The *Root* provider authenticates for the GeoServer *Root account* and does not use a user/group service. The *Username/password* provider is the default provider and relays username and password credentials to a user/group service.

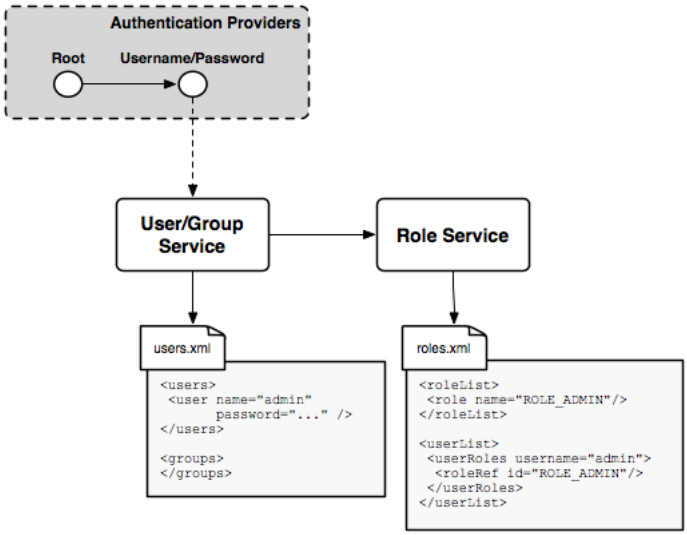


Fig. 13.42: *Default GeoServer security configuration*

A single user/group service, which persist the user database as XML, is present. The database contains a single user named `admin` and no groups. Similarly, the role service persists the role database as XML. By default, this contains a single role named `ADMIN`, which is associated with the `admin` user. The `ADMIN` role is mapped to the `ROLE_ADMINISTRATOR` role and as a result, the `admin` user is associated with system administrator role during role calculation.

13.3 Authentication

There are three sets of GeoServer resources involved in authentication:

- The *Web administration interface* (also known as web admin)
- *OWS* services (such as WFS and WMS)
- *REST* services

The following sections describe how each set of GeoServer resources administers authentication. To configure the authentication settings and providers, please see the section on *Authentication* in the *Web administration interface*.

13.3.1 Authentication chain

Understanding the **authentication chain** helps explain how GeoServer authentication works. The authentication chain processes requests and applies certain authentication mechanisms. Examples of authentication mechanisms include:

- **Username/password**—Performs authentication by looking up user information in an external user database
- **Browser cookie**—Performs authentication by recognizing previously sent browser cookies (also known as "Remember Me")
- **LDAP**—Performs authentication against an LDAP database

- **Anonymous**—Essentially performs no authentication and allows a request to proceed without any credentials

Multiple authentication mechanisms may be active within GeoServer at a given time. The following figure illustrates the flow of a generic request.

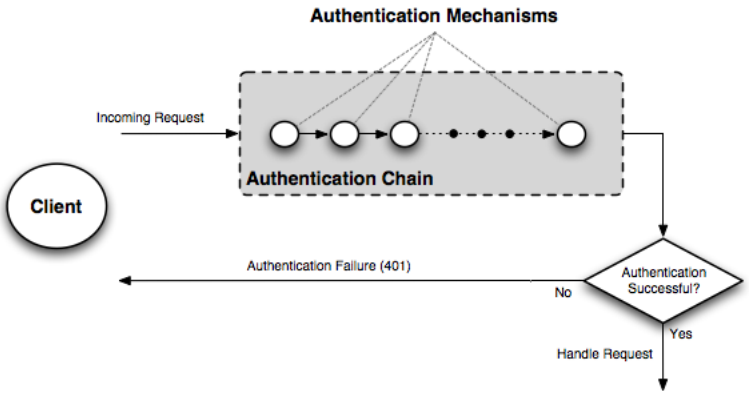


Fig. 13.43: *Flow of a request through the authentication system*

Before dispatching a request to the appropriate service or handler, GeoServer first filters the request through the authentication chain. The request is passed to each mechanism in the chain in order, and each is given the chance to authenticate the request. If one of the mechanisms in the chain is able to successfully authenticate, the request moves to normal processing. Otherwise the request is not routed any further and an authorization error (usually a HTTP 401) is returned to the user.

Filter chain and provider chain

In the case of GeoServer, the authentication chain is actually made up of two chains: a **filter chain**, which determine if further authentication of a request is required, and a **provider chain**, which performs the actual authentication.

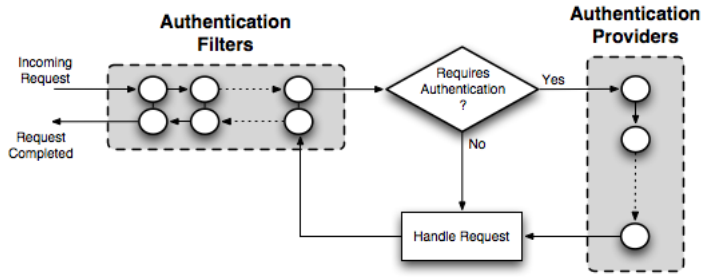


Fig. 13.44: *Detail of authentication chain, showing filter chain and provider chain*

The filter chain performs a variety of tasks, including:

- Gathering user credentials from a request, for example from Basic and Digest Authentication headers
- Handling events such as ending the session (logging out), or setting the "Remember Me" browser cookie

- Performing session integration, detecting existing sessions and creating new sessions if necessary
- Invoking the authentication provider chain to perform actual authentication

The filter chain is actually processed twice, before and after the request is handled.

The provider chain is concerned solely with performing the underlying authentication of a request. It is invoked by the filter chain when a filter determines that authentication is required.

Filter chain by request type

A different **filter chain** can be applied to each different type of request in GeoServer. This happens because the administrator can configure a list of different filter chains and a matching rule for each of them. Only the first matching chain of the configured ordered list will be applied to any given request.

Matching rules can be applied to:

- HTTP Method (GET, POST, etc.)
- one or more ANT patterns for the path section of the request (e.g /wms/**); if more than one pattern (comma delimited) is specified, any of them will match
- an optional regular expression to match parameters on the query string, for one or more of the specified ANT pattern; if the path matches, also the query string is checked for matching; the regular expression can be specified after the ANT pattern, with a pipe (|) separator

ANT Patterns support the following wildcards:

- ? matches one character
- * matches zero or more characters
- ** matches zero or more 'directories' in a path

Query String regular expressions will match the full query string (^ and $ terminators are automatically appended), so to match only part of it, remember to prefix and postfix the expression with .* (e.g. .*request=getcapabilities.*)

Examples of rules (ANT patterns and query string regular expressions)

Pattern	Description
/wms, /wms/**	simple ANT pattern
/wms|.*request=GetMap.*	ANT pattern and querystring regex to match one parameter
/wms|(?=.*request=getmap)(?=.*format=image/png).*	ANT pattern and querystring regex to match two parameters in any order
/wms|(?=.*request=getmap)(?!.*format=image/png).*	ANT pattern and querystring regex to match one parameters and be sure another one is not matched

13.3.2 Authenticating to the Web Admin Interface

The method of authenticating to the *Web administration interface* application is typical of most web applications that provide login capabilities. The application is based primarily on form-based authentication, in which a user authenticates through a form in a web browser. Upon successful authentication a session is created on the server, eliminating the need for a user to repeat the login process for each page they wish to

access. An optional "Remember Me" setting is also supported which will store authentication information in a client-side cookie to allow the user to bypass the form-based authentication after the initial session has timed out.

The typical process of authentication is as follows:

1. User visits the home page of the web admin for the very first time, so neither a session or "Remember Me" cookie is present. In this case, the user is anonymously authenticated.
2. User accesses a secured page and is presented with a login form.
3. Upon successful login a session is created. Depending on the privileges of the account used to log in, the user will either be directed to the requested page or be redirected back to the home page.
4. Upon subsequent requests to secured pages, the user is authenticated via browser session until the session expires or the user logs out.

Examples

The following shows the default configuration of the authentication chain for the web admin.

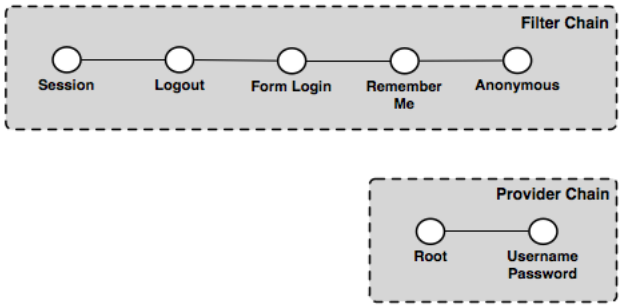


Fig. 13.45: *GeoServer authentication chain, with filter and provider chains*

In this example the filter chain is made up of the following filters:

- **Session**—Handles session integration, recognizing existing sessions and creating new sessions on demand
- **Logout**—Handles ending sessions (user logout)
- **Form login**—Handles form logins
- **Remember Me**—Handles "Remember Me" authentication, reading when the flag is set on a form login, creating the appropriate cookie, and recognizing the cookie on future requests
- **Anonymous**—Handles anonymous access

The provider chain is made up of two providers:

- **Root**—The *Root account* has a special "super user" provider. As this account is rarely used, this provider is rarely invoked.
- **Username/password**—Performs username/password authentication against a user database.

To following example requests illustrate how the elements of the various chains work.

First time visit

This example describes the process when a user visits the home page of the web admin for the first time.

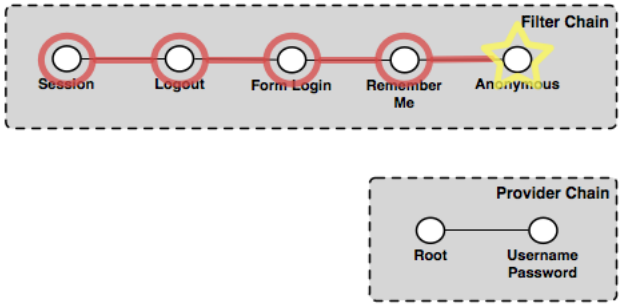


Fig. 13.46: *Authentication chain for a first time visit from a user*

The first filter to execute is the *Session* filter. It checks for an existing session, but finds none, so processing continues to the next filter in the chain. The *Logout* filter checks for the case of a user logging out, which also is not the case, so processing continues. The *Form login* filter checks for a form login, and also finds none. The *Remember Me* filter determines if this request can be authenticated from a previous session cookie, but in this case it cannot. The final filter to execute is the *Anonymous* filter which checks if the user specified any credentials. In this case the user has not provided any credentials, so the request is authenticated anonymously. Since no authentication is required to view the home page, the provider chain is not invoked.

The last response to the request directs the user to the home page.

User logs on

This examples describes the process invoked when a user logs on to the web admin via the login form.

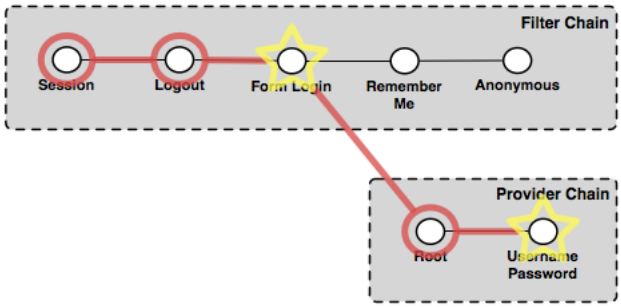


Fig. 13.47: *Authentication chain for a user logging in*

The *Session* filter finds no existing session, and processing continues. The *Logout* filter checks for a logout request, finds none, and continues. The *Form login* filter recognizes the request as a form login and begins the authentication process. It extracts the username and password from the request and invokes the provider chain.

In the provider chain, the *Root* provider checks for the root account login, but doesn't find it so processing continues to the next provider. The *Username/password* provider checks if the supplied credentials are valid. If they are valid the authentication succeeds, user is redirected to the home page and is considered to be

logged on. During the post-processing step the *Session* filter recognizes that a successful authentication has taken place and creates a new session.

If the credentials are invalid, the user will be returned to the login form page and asked to try again.

User visits another page

This example describes the process invoked when a user who is already logged on visits another page in the web admin.

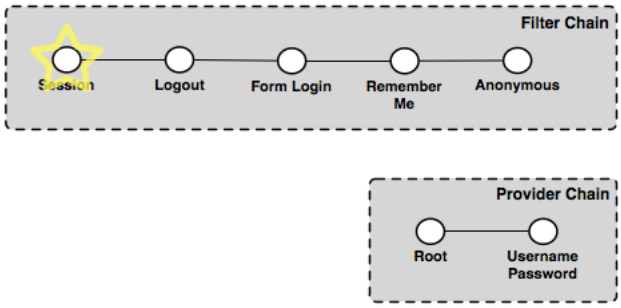


Fig. 13.48: *Authentication chain for a user visiting another page after logging in*

The *Session* filter executes and finds an existing session that is still valid. The session contains the authentication details and no further chain processing is required. The response is the page requested by the user.

User returns after session time out

This example describes the process invoked when a user returns to the web admin after the previously created session has timed out.

A session will time out after a certain period of time. When the user returns to the web admin, this becomes essentially the same chain of events as the user visiting the web app for the first time (as described previously). The chain proceeds to the *Anonymous* filter that authenticates anonymously. Since the page requested is likely to be a page that requires authentication, the user is redirected to the home page and is not logged on.

User logs on with "Remember Me" flag set

This example describes the process for logging on with the "Remember Me" flag set.

The chain of events for logging on with "Remember Me" set is identical to the process for when the flag is not set, except that after the successful authentication the *Form login* filter recognizes the "Remember Me" flag and triggers the creation of the browser cookie used to persist the authentication information. The user is now logged on and is directed to the home page.

User returns after session time out (with "Remember Me")

This example describes the process invoked when the user returns to the web admin after a period of inactivity, while the "Remember Me" flag is set.

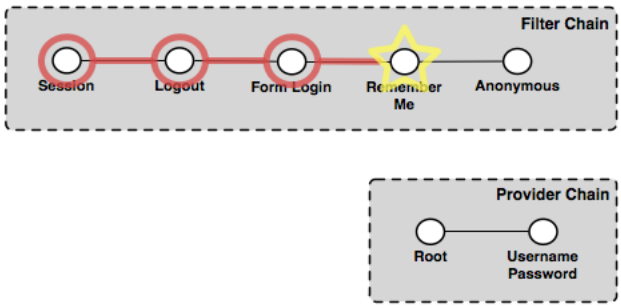


Fig. 13.49: *Authentication chain for a user returning after session time out with the "Remember Me" flag*

Even though the "Remember Me" flag is set, the user's session on the server will still time out as normal. As such, the chain proceeds accordingly through the filters, starting with the *Session* filter, which finds no valid session. The *Logout* and *Form login* filters do not apply here. The *Remember Me* filter recognizes the browser cookie and is able to authenticate the request. The user is directed to whatever page was accessed and remains logged on.

13.3.3 Authentication to OWS and REST services

OWS and REST services are stateless and have no inherent awareness of "session", so the authentication scheme for these services requires the client to supply credentials on every request. That said, "session integration" is supported, meaning that if a session already exists on the server (from a concurrent *authenticated web admin session*) it will be used for authentication. This scheme allows GeoServer to avoid the overhead of session creation for OWS and REST services.

The default GeoServer configuration ships with support for HTTP Basic authentication for services.

The typical process of authentication is as follows:

1. User makes a service request without supplying any credentials
2. If the user is accessing an unsecured resource, the request is handled normally
3. If the user is accessing a secured resource:

- An HTTP 401 status code is sent back to the client, typically forcing the client to prompt for credentials.
- The service request is then repeated with the appropriate credentials included, usually in the HTTP header as with Basic Authentication.
- If the user has sufficient privileges to access the resource the request is handled normally, otherwise, a HTTP 404 status code is returned to the client.

4. Subsequent requests should include the original user credentials

Examples

The following describes the authentication chain for an OWS service:

In this example the filter chain consists of three filters:

- **Session**—Handles "session integration", recognizing existing sessions (but *not* creating new sessions)
- **Basic Auth**—Extracts Basic Authentication credentials from request HTTP header
- **Anonymous**—Handles anonymous access

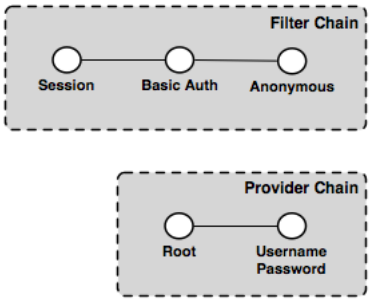


Fig. 13.50: *The OWS service authentication chain*

The provider chain is made up of two providers:

- **Root**—*Root account* has a special "super user" provider. As this account is rarely used, this provider is rarely invoked.
- **Username/password**—Performs username/password authentication against a user database

To illustrate how the elements of the various chains work, here are some example OWS requests.

Anonymous WMS GetCapabilities request

This example shows the process for when a WMS client makes an anonymous GetCapabilities request.

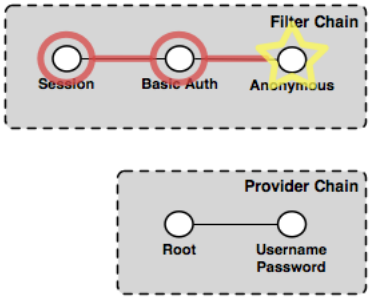


Fig. 13.51: *Authentication chain for a WMS client making an anonymous GetCapabilities request*

The *Session* filter looks for an existing session, but finds none, so processing continues. The *Basic Auth* filter looks for the Basic Authorization header in the request, but as the request is anonymous, the filter finds none. Finally, the *Anonymous* filter executes and authenticates the request anonymously. Since GetCapabilities is a "discovery" operation it is typically not locked down, even on a secure server. Assuming this is the case here, the anonymous request succeeds, returning the capabilities response to the client. The provider chain is not invoked.

Anonymous WMS GetMap request for a secured layer

This example shows the process invoked when a WMS client makes an anonymous GetMap request for a secured layer

The chain executes exactly as described above. The *Session* filter looks for an existing session, but finds none, so processing continues. The *Basic Auth* filter looks for the Basic Authorization header in the request, but as

the request is anonymous, the filter finds none. Finally, the *Anonymous* filter executes and authenticates the request anonymously. However, in this case the layer being accessed is a secured resource, so the handling of the GetMap request fails. The server returns an exception accompanied with a HTTP 401 status code, which usually triggers the client presenting the user with a login dialog.

WMS GetMap request with user-supplied credentials

This example shows the process invoked when a WMS client gathers credentials from the user and reissues the previous request for a secured layer.

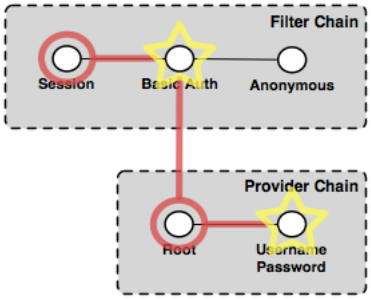


Fig. 13.52: *Authentication chain for a WMS client making a GetMap request with user-supplied credentials*

The *Session* filter executes as described above, and does nothing. The *Basic Auth* filter finds the authorization header in the request, extracts the credentials for it, and invokes the provider chain. Processing moves to the *Username/password* provider that does the actual authentication. If the credentials have the necessary privileges to access the layer, the processing of the request continues normally and the GetMap request succeeds, returning the map response. If the credentials are not sufficient, the HTTP 401 status code will be supplied instead, which may again trigger the login dialog on the client side.

13.3.4 Authentication providers

The following authentication providers are available in GeoServer:

- Authentication of a username/password against a *user/group service*
- Authentication against an LDAP server
- Authentication by connecting to a database through JDBC

Username/password authentication

Username and password authentication is the default authentication provider. It uses a *user/group service* to authenticate.

The provider simply takes the username/password from an incoming request (such as a Basic Authentication request), then loads the user information from the user/group service and verifies the credentials.

LDAP authentication

The LDAP authentication provider allows for authentication against a Lightweight Directory Access Protocol (LDAP) server. The provider takes the username/password from the incoming request and attempts to connect to the LDAP server with those credentials.

Note: Currently only LDAP Bind authentication is supported.

Role assignment

The LDAP provider offers two options for role assignment for authenticated users:

- Convert the user's LDAP groups into roles
- Employ a user/group service

The following LDAP database will illustrate the first option:

```
dn: ou=people,dc=acme,dc=com
objectclass: organizationalUnit
ou: people

dn: uid=bob,ou=people,dc=acme,dc=com
objectclass: person
uid: bob

dn: ou=groups,dc=acme,dc=com
objectclass: organizationalUnit
ou: groups

dn: cn=workers,ou=groups,dc=acme,dc=com
objectclass: groupOfNames
cn: users
member: uid=bob,ou=people,dc=acme,dc=com
```

The above scenario defines a user with the `uid` of `bob`, and a `group` named `workers` of which `bob` is a member. After authentication, `bob` will be assigned the role `ROLE_WORKERS`. The role name is generated by concatenating `ROLE_` with the name of the group in upper case.

Note: When the LDAP server doesn't allow searching in an anonymous context, the bindBeforeGroupSearch option should be enabled to avoid errors.

In the case of using a *user/group service*, the user/group service is queried for the user following authentication, and the role assignment is performed by both the user/group service and the active *role service*. When using this option, any password defined for the user in the user/group service database is ignored.

Secure LDAP connections

There are two ways to create a secure LDAP connection with the server. The first is to directly specify a secure connection by using the **ldaps** protocol as part of the *Server URL*. This typically requires changing the connection port to **port 636** rather than 389.

The second method involves using **STARTTLS** (Transport Layer Security) to negotiate a secure connection over a non-secure one. The negotiation takes place over the non-secure URL using the "ldap" protocol on port 389. To use this option, the *Use TLS* flag must be set.

Warning: Using TLS for connections will prevent GeoServer from being able to pool LDAP connections. This means a new LDAP connection will be created and destroyed for each authentication, resulting in loss of performance.

JDBC authentication

The JDBC authentication provider authenticates by connecting to a database over JDBC.

The provider takes the username/password from the incoming request and attempts to create a database connection using those credentials. Optionally the provider may use a *user/group service* to load user information after a successful authentication. In this context the user/group service will not be used for password verification, only for role assignment.

Note: To use the user/group service for password verification, please see the section on *Username/password authentication*.

13.4 Passwords

Passwords are a central aspect of any security system. This section describes how GeoServer handles passwords.

13.4.1 Password encryption

A GeoServer configuration stores two types of passwords:

- Passwords for **user accounts** to access GeoServer resources
- Passwords used internally for **accessing external services** such as databases and cascading OGC services

As these passwords are typically stored on disk it is strongly recommended that they be encrypted and not stored as human-readable text. GeoServer security provides four schemes for encrypting passwords: **empty**, **plain text**, **Digest**, and **Password-based encryption (PBE)**.

The password encryption scheme is specified as a global setting that affects the encryption of passwords used for external resources, and as an encryption scheme for each *user/group service*. The encryption scheme for external resources has to be be *reversible*, while the user/group services can use any scheme.

Empty

The scheme is not reversible. Any password is encoded as an empty string, and as a consequence it is not possible to recalculate the plain text password. This scheme is used for user/group services in combination with an authentication mechanism using a back end system. Examples are user name/password authentication against a LDAP server or a JDBC database. In these scenarios, storing passwords locally to GeoServer does not make sense.

Plain text

Plain text passwords provide no encryption at all. In this case, passwords are human-readable by anyone who has access to the file system. For obvious reasons, this is not recommended for any but the most basic test server. A password `mypassword` is encoded as `plain:mypassword`, the prefix uniquely describing the algorithm used for encoding/decoding.

Digest

Digest encryption is not reversible. It applies, 100,000 times through an iterative process, a SHA-256 cryptographic hash function to passwords. This scheme is "one-way" in that it is virtually impossible to reverse and obtain the original password from its hashed representation. Please see the section on *Reversible encryption* for more information on reversibility.

To protect from well known attacks, a random value called a salt is added to the password when generating the key. For each digesting, a separate random salt is used. Digesting the same password twice results in different hashed representations.

As an example, the password `geoserver` is digested to `digest1:YgaweuS60t+mJNobGlf9hzUC6g7gGTtPEu0TlnUxFl`. `digest1` indicates the usage of digesting. The hashed representation and the salt are base 64 encoded.

Password-based encryption

Password-based encryption (PBE) normally employs a user-supplied password to generate an encryption key. This scheme is reversible. A random salt described in the previous section is used.

Note: The system never uses passwords specified by users because these passwords tend to be weak. Passwords used for encryption are generated using a secure random generator and stored in the GeoServer key store. The number of possible passwords is 2^260 .

GeoServer supports two forms of PBE. **Weak PBE** (the GeoServer default) uses a basic encryption method that is relatively easy to crack. The encryption key is derived from the password using MD5 1000 times iteratively. The encryption algorithm itself is DES (Data Encryption Standard). DES has an effective key length of 56 bits, which is not really a challenge for computer systems in these days.

Strong PBE uses a much stronger encryption method based on an AES 256-bit algorithm with CBC. The key length is 256 bit and is derived using SHA-256 instead of MD5. Using Strong PBE is highly recommended.

As an example, the password `geoserver` is encrypted to `crypt1:KWhO7jrTz/Gi0oTQRKsVeCmWIZY5VZaD`. `crypt1` indicates the usage of Weak PBE. The prefix for Strong PBE is `crypt2`. The ciphertext and the salt are base 64 encoded.

Note: Strong PBE is not natively available on all Java virtual machines and may require *Installing Unlimited Strength Jurisdiction Policy Files*

Reversible encryption

Password encryption methods can be **reversible**, meaning that it is possible (and desirable) to obtain the plain-text password from its encrypted version. Reversible passwords are necessary for database connections or external OGC services such as *cascading WMS* and *cascading WFS*, since GeoServer must be able

to decode the encrypted password and pass it to the external service. Plain text and PBE passwords are reversible.

Non-reversible passwords provide the highest level of security, and therefore should be used for user accounts and wherever else possible. Using password digesting is highly recommended, the installation of the unrestricted policy files is not required.

13.4.2 Secret keys and the keystore

For a reversible password to provide a meaningful level of security, access to the password must be restricted in some way. In GeoServer, encrypting and decrypting passwords involves the generation of secret shared keys, stored in a typical Java *keystore*. GeoServer uses its own keystore for this purpose named `geoserver.jceks` which is located in the `security` directory in the GeoServer data directory. This file is stored in the JCEKS format rather than the default JKS. JKS does not support storing shared keys.

The GeoServer keystore is password protected with a *Master password*. It is possible to access the contents of the keystore with external tools such as keytool. For example, this following command would prompt for the master password and list the contents of the keystore:

```
$ keytool -list -keystore geoserver.jceks -storetype "JCEKS"
```

13.4.3 Master password

It is also possible to set a **master password** for GeoServer. This password serves two purposes:

- Protect access to the *keystore*
- Protect access to the GeoServer *Root account*

By default, the master password is generated and stored in a file named `security/masterpw.info` using plain text. When upgrading from an existing GeoServer data directory (versions 2.1.x and lower), the algorithm attempts to figure out the password of a user with the role `ROLE_ADMINISTRATOR`. If such a password is found and the password length is 8 characters at minimum, GeoServer uses this password as master password. Again, the name of the chosen user is found in `security/masterpw.info`.

Warning: The file `security/masterpw.info` is a security risk. The administrator should read this file and verify the master password by logging on GeoServer as the `root` user. On success, this file should be removed.

Warning: The first thing an Administrator of the System should do is dump the Master Password generated by GeoServer, store it somewhere not accessible by anyone, and delete `security/masterpw.info` or whatever file you dumped the password to.

Refer to *Active master password provider* for information on how to change the master password.

Note: By default login to the Admin GUI and REST APIs using the Master Password is disabled. In order to enable it you will need to manually change the Master Password Provider config.xml, usually located in `security/masterpw/default/config.xml`, by adding the following statement:

```
``<loginEnabled>true</loginEnabled>``
```

13.4.4 Password policies

A password policy defines constraints on passwords such as password length, case, and required mix of character classes. Password policies are specified when adding *User/group services* and are used to constrain passwords when creating new users and when changing passwords of existing users.

Each user/group service uses a password policy to enforce these rules. The default GeoServer password policy implementation supports the following optional constraints:

- Passwords must contain at least one number
- Passwords must contain at least one upper case letter
- Passwords must contain at least one lower case letter
- Password minimum length
- Password maximum length

13.5 Root account

The highly configurable nature of GeoServer security may result in an administrator inadvertently disrupting normal authentication, essentially disabling all users including administrative accounts. For this reason, the GeoServer security subsystem contains a **root account** that is always active, regardless of the state of the security configuration. Much like its UNIX-style counterpart, this account provides "super user" status, and is meant to provide an alternative access method for fixing configuration issues.

The user name for the root account is `root`. Its name cannot be changed and the password for the root account is the *Master password*.

13.6 Service Security

GeoServer supports access control at the service level, allowing for the locking down of service operations to only authenticated users who have been granted a particular role. There are two main categories of services in GeoServer. The first is *OWS services* such as WMS and WFS. The second are RESTful services, such as the GeoServer *REST*.

Note: Service-level security and *Layer security* cannot be combined. For example, it is not possible to specify access to a specific OWS service only for one specific layer.

13.6.1 OWS services

OWS services support setting security access constraints globally for a particular service, or to a specific operation within that service. A few examples include:

- Securing the entire WFS service so only authenticated users have access to all WFS operations.
- Allowing anonymous access to read-only WFS operations such as GetCapabilities, but securing write operations such as Transaction.
- Disabling the WFS service in effect by securing all operations and not applying the appropriate roles to any users.

OWS service security access rules are specified in a file named `services.properties`, located in the `security` directory in the GeoServer data directory. The file contains a list of rules that map service operations to defined roles. The syntax for specifying rules is as follows:

```
<service>.<operation|*>=<role>[,<role2>,...]
```

The parameters include:

- `[]`—Denotes optional parameters
- `|`—Denotes "or"
- `service`—Identifier of an OGC service, such as `wfs`, `wms`, or `wcs`
- `operation`—Any operation supported by the service, examples include `GetFeature` for WFS, `GetMap` for WMS, `*` for all operations
- `role[,role2,...]`—List of predefined role names

Note: It is important that roles specified are actually linked to a user, otherwise the whole service/operation will be accessible to no one except for the *Root account*. However in some cases this may be the desired effect.

The default service security configuration in GeoServer contains no rules and allows any anonymous user to access any operation of any service. The following are some examples of desired security restrictions and the corresponding rules.

Securing the entire WFS service

This rule grants access to any WFS operation only to authenticated users that have been granted the `ROLE_WFS` role:

```
wfs.*=ROLE_WFS
```

Anonymous WFS access only for read-only operations

This rule grants anonymous access to all WFS operations (such as GetCapabilities and GetFeature) but restricts Transaction requests to authenticated users that have been granted the `ROLE_WFS_WRITE` role:

```
wfs.Transaction=ROLE_WFS_WRITE
```

Securing data-accessing WFS operations and write operations

Used in conjunction, these two rules grant anonymous access to GetCapabilities and DescribeFeatureType, forcing the user to authenticate for the GetFeature operation (must be granted the `ROLE_WFS_READ` role), and to authenticate to perform transactions (must be granted the `ROLE_WFS_WRITE` role:

```
wfs.GetFeature=ROLE_WFS_READ
wfs.Transaction=ROLE_WFS_WRITE
```

Note this example does not specify whether a user accessing Transactions would also have access to GetFeature.

13.6.2 REST services

In addition to providing the ability to secure OWS services, GeoServer also allows for the securing of RESTful services.

REST service security access rules are specified in a file named `rest.properties`, located in the `security` directory of the GeoServer data directory. This file contains a list of rules mapping request URIs to defined roles. The rule syntax is as follows:

```
<uriPattern>;<method>[,<method>,...]=<role>[,<role>,...]
```

The parameters include:

- `[]`—Denote optional parameters
- `uriPattern`—The *ant pattern* that matches a set of request URIs
- `method`—HTTP request method, one of `GET`, `POST`, `PUT`, `POST`, `DELETE`, or `HEAD`
- `role`—Name of a predefined role. The wildcard `*` is used to indicate all users, including anonymous users.

Note:

- URI patterns should account for the first component of the rest path, usually `rest` or `api`
- `method` and `role` lists should **not** contain any spaces

Ant patterns

Ant patterns are commonly used for pattern matching directory and file paths. The following examples provide some basic instructions. The Apache ant user manual contains more sophisticated use cases.

These examples are specific to GeoServer *REST*, but any RESTful GeoServer service could be configured in the same manner.

Disabling anonymous access to services

The most secure of configurations is one that forces any request, REST or otherwise, to be authenticated. The following will lock down access to all requests to users that are granted the `ROLE_ADMINISTRATOR` role:

```
/**;GET,POST,PUT,DELETE=ROLE_ADMINISTRATOR
```

A less restricting configuration locks down access to operations under the path `/rest` to users granted the `ROLE_ADMINISTRATOR` role, but will allow anonymous access to requests that fall under other paths (for example `/api`):

```
/rest/**;GET,POST,PUT,DELETE=ROLE_ADMINISTRATOR
```

Allowing anonymous read-only access

The following configuration grants anonymous access when the `GET` method is used, but forces authentication for a `POST`, `PUT`, or `DELETE` method:

```
/**;GET=IS_AUTHENTICATED_ANONYMOUSLY
/**;POST,PUT,DELETE=TRUSTED_ROLE
```

Securing a specific resource

The following configuration forces authentication for access to a particular resource (in this case the `states` feature type):

```
/rest/**/states*;GET=TRUSTED_ROLE
/rest/**;POST,PUT,DELETE=TRUSTED_ROLE
```

The following secures access to a set of resources (in this case all data stores).:

```
/rest/**/datastores/*;GET=TRUSTED_ROLE
/rest/**/datastores/*.*;GET=TRUSTED_ROLE
/rest/**;POST,PUT,DELETE=TRUSTED_ROLE
```

Note the trailing wildcards `/*` and `/*.*`.

13.7 Layer security

GeoServer allows access to be determined on a per-layer basis.

Note: Layer security and *Service Security* cannot be combined. For example, it is not possible to specify access to a specific OWS service, only for one specific layer.

Providing access to layers is linked to *roles*. Layers and roles are linked in a file called `layers.properties`, which is located in the `security` directory in your GeoServer data directory. The file contains the rules that control access to workspaces and layers.

Note: The default layers security configuration in GeoServer allows any anonymous user to read data from any layer but only allows admin users to edit data.

13.7.1 Rules

The syntax for a layer security rule can follow three different patterns (`[]` denotes optional parameters):

```
globalLayerGroup.permission=role[,role2,...]
workspace.layer.permission=role[,role2,...]
workspace.workspaceLayerGroup.permission=role[,role2,...]
```

The parameters include:

- `globalLayerGroup`— Name of a global layer group (one without workspace associated to it).
- `workspace`— Name of the workspace. The wildcard `*` is used to indicate all workspaces.
- `layer`— Name of a resource (featuretype/coverage/etc...). The wildcard `*` is used to indicate all layers.
- `workspaceLayerGroup`— Name of a workspace specific layer group.

- `permission`— Type of access permission/mode.
  - `r`—Read access
  - `w`—Write access
  - `a`—Admin access

  See *Access modes* for more details.
- `role[,role2,...]` is the name(s) of predefined roles. The wildcard `*` is used to indicate the permission is applied to all users, including anonymous users.

Note: If a workspace or layer name is supposed to contain dots, they can be escaped using double backslashes (`\\`). For example, if a layer is named `layer.with.dots` the following syntax for a rule may be used:

```
topp.layer\\.with\\.dots.r=role[,role2,...]
```

General rules for layer security:

- Each entry must have a unique combination of workspace, layer, and permission values.
- If a permission at the global level is not specified, global permissions are assumed to allow read/write access.
- If a permission for a workspace is not specified, it inherits permissions from the global specification.
- If a permission for a layer is not specified, it inherits permissions from its workspace specification in all protocols except WMS (where layer groups rules play a role, see below).
- If a user belongs to multiple roles, the **least restrictive** permission they inherit will apply.

For WMS, layers will be also secured by considering their containing layer groups. In particular:

- Rules with *Single* layer groups only affect the group itself, but not its contents. *Single* mode is considered just an alias for a list of layers, with no containment.
- Rules with other types of groups (*Named tree*, *Container tree*, *Earth Observation tree*) also affect contained layers and nested layer groups. If the group is not accessible, the layers and groups contained in that group will not be accessible either.. The only exception is when another layer group which is accessible contains the same layer or nested group, in that case the layers they will show up under the allowed groups.
- Workspace rules gets precedence over global layer group ones when it comes to allow access to layers.
- Layer rules get precedence over all layer group rules when it comes to allow access to layers.

The following tables summarizes the layer group behavior depending on whether they are used in a public or secured environment:

Mode	Public behavior	Restricted behavior
Single	Looks like a stand-alone layer, can be requested directly and acts as an alias for a layer list. Layers are also visible at root level	Does not control layer access at all
Opaque container	Looks like a stand-alone layer, can be requested directly and acts as an alias for a layer list. Layers in it are not available for WMS requests	Same as public behavior
Named tree	Contained layers are visible as children in the capabilities document, the name can be used as a shortcut to request all layers.	Restricting access to the group restricts also the contained layers, unless another "tree" group allows access to the same layer
Container tree	Same as "named tree", but does not have a name published in the capabilities document and thus cannot be requested directly	Same as "named tree"

13.7.2 Catalog Mode

The `layers.properties` file may contain a further directive that specifies how GeoServer will advertise secured layers and behave when a secured layer is accessed without the necessary privileges. The parameter is `mode` and is commonly referred to as the "catalog mode".

The syntax is:

```
mode=option
```

`option` may be one of three values:

Option	Description
`hide`	*(Default)* Hides layers that the user does not have read access to, and behaves as if a layer is read only if the user does not have write permissions. The capabilities documents will not contain the layers the current user cannot access. This is the highest security mode. As a result, it may not work very well with clients such as uDig or Google Earth.
`challenge`	Allows free access to metadata, but any attempt at accessing actual data is met by a HTTP 401 code (which forces most clients to show an authentication dialog). The capabilities documents contain the full list of layers. DescribeFeatureType and DescribeCoverage operations work successfully. This mode works fine with clients such as uDig or Google Earth.
`mixed`	Hides the layers the user cannot read from the capabilities documents, but triggers authentication for any other attempt to access the data or the metadata. This option is useful if you don't want the world to see the existence of some of your data, but you still want selected people to who have data access links to get the data after authentication.

13.7.3 Access modes

The access mode defines what level of access should be granted on a specific workspace/layer to a particular role. There are three types of access mode:

- `r`—**Read mode** (read data from a workspace/layer)
- `w`—**Write mode** (write data to a workspace/layer)
- `a`—**Admin mode** (access and modify the configuration of a workspace/layer)

Some notes on the above access modes:

- Write does not imply Read, but Admin implies both Write *and* Read.
- Read and Write apply to the data of a layer, while Admin applies to the configuration of a layer.
- As Admin mode only refers to the configuration of the layer, it is not required for any OGC service request.

Note: Currently, it is possible to assign Admin permission only to an entire workspace, and not to specific layers.

13.7.4 Examples

The following examples illustrate some possible layer restrictions and the corresponding rules.

Protecting a single workspace and a single layer

The following example demonstrates how to configure GeoServer as a primarily a read-only server:

```
*.*.r=*
*.*.w=NO_ONE
private.*.r=TRUSTED_ROLE
private.*.w=TRUSTED_ROLE
topp.congress_district.w=STATE_LEGISLATORS
```

The mapping of roles to permissions is as follows:

Role	private.*	topp.*	topp.congress_district	(all other workspaces)
NO_ONE	(none)	w	(none)	w
TRUSTED_ROLE	r/w	r	r	r
STATE_LEGISLATURES	(none)	r	r/w	r
(All other users)	r	r	r	r

Locking down GeoServer

The following example demonstrates how to lock down GeoServer:

```
*.*.r=TRUSTED_ROLE
*.*.w=TRUSTED_ROLE
topp.*.r=*
army.*.r=MILITARY_ROLE,TRUSTED_ROLE
army.*.w=MILITARY_ROLE,TRUSTED_ROLE
```

The mapping of roles to permissions is as follows:

Role	topp.*	army.*	(All other workspaces)
TRUSTED_ROLE	r/w	r/w	r/w
MILITARY_ROLE	r	r/w	(none)
(All other users)	r	(none)	(none)

Providing restricted administrative access

The following provides administrative access on a single workspace to a specific role, in additional to the full administrator role:

```
*.*.a=ROLE_ADMINISTRATOR
topp.*.a=ROLE_TOPP_ADMIN,ROLE_ADMINISTRATOR
```

Managing multi-level permissions

The following example demonstrates how to configure GeoServer with global-, workspace–, and layer-level permissions:

```
*.*.r=TRUSTED_ROLE
*.*.w=NO_ONE
topp.*.r=*
topp.states.r=USA_CITIZEN_ROLE,LAND_MANAGER_ROLE,TRUSTED_ROLE
topp.states.w=NO_ONE
topp.poly_landmarks.w=LAND_MANAGER_ROLE
topp.military_bases.r=MILITARY_ROLE
topp.military_bases.w=MILITARY_ROLE
```

The mapping of roles to permissions is as follows:

Role	topp.states	topp.poly_landmarks	topp.military_bases	topp.(all other layers)	(All other workspaces)
NO_ONE	w	r	(none)	w	w
TRUSTED_ROLE	r	r	(none)	r	r
MILITARY_ROLE	(none)	r	r/w	r	(none)
USA_CITIZEN_ROLE	r	r	(none)	r	(none)
LAND_MANAGER_ROLE	r	r/w	(none)	r	(none)
(All other users)	(none)	r	(none)	r	(none)

Note: The entry `topp.states.w=NO_ONE` is not required because this permission would be inherited from the global level (the entry `*.*.w=NO_ONE`).

Invalid configuration

The following examples are invalid because the workspace, layer, and permission combinations are not unique:

```
topp.state.rw=ROLE1
topp.state.rw=ROLE2,ROLE3
```

Security by layer group in WMS

To clarify, lets assume the following starting situation, in which all layers and groups are visible:

```
root
+- namedTreeGroupA
|   |   ws1:layerA
|      ws2:layerB
+- namedTreeGroupB
|   |   ws2:layerB
|      ws1:layerC
+- layerD
+- singleGroupC (contains ws1:layerA and layerD when requested)
```

Here are a few examples of how the structure changes based on different security rules.

- Denying access to `namedTreeGroupA` by:

  ```
  namedTreeGroupA.r=ROLE_PRIVATE
  ```

  Will give the following capabilities tree to anonymous users:

  ```
  root
  +- namedTreeGroupB
  |   |   ws2:layerB
  |      ws1:layerC
  +- layerD
  +- singleGroupC (contains only layerD when requested)
  ```

- Denying access to `namedTreeGroupB` by

  ```
  namedTreeGroupB.r=ROLE_PRIVATE
  ```

  Will give the following capabilities tree to anonymous users:

  ```
  root
  +- namedTreeGroupA
  |   |   ws1:layerA
  |      ws2:layerB
  +- layerD
  +- singleGroupC (contains ws1:layerA and layerD when requested)
  ```

- Denying access to `singleGroupC` by:

  ```
  singleGroupC.r=ROLE_PRIVATE
  ```

  Will give the following capabilities tree to anonymous users:

  ```
  root
  +- namedTreeGroupA
  |   |   ws1:layerA
  |      ws2:layerB
  +- namedTreeGroupB
  |   |   ws2:layerB
  |      ws1:layerC
  +- layerD
  ```

- Denying access to everything, but allowing explicit access to namedTreeGroupA by:

  ```
  nameTreeGroupA.r=*
  *.*.r=PRIVATE
  *.*.w=PRIVATE
  ```

Will give the following capabilities tree to anonymous users:

```
root
+- namedTreeGroupA
    |   ws1:layerA
       ws2:layerB
```

- Denying access to `nameTreeA` and `namedTreeGroupB` but explicitly allowing access to `ws1:layerA`:

```
namedTreeGroupA.r=ROLE_PRIVATE
namedTreeGroupB.r=ROLE_PRIVATE
ws1.layerA.r=*
```

Will give the following capabilities tree to anonymous users (notice how ws1:layerA popped up to the root):

```
root
+- ws1:layerA
+- layerD
```

- Denying access to `nameTreeA` and `namedTreeGroupB` but explicitly allowing all layers in ws2 (a workspace rules overrides global groups ones):

```
namedTreeGroupA.r=ROLE_PRIVATE
namedTreeGroupB.r=ROLE_PRIVATE
ws2.*.r=*
```

Will give the following capabilities tree to anonymous users (notice how ws1:layerB popped up to the root):

```
root
+- ws2:layerB
+- layerD
+- singleGroupC
```

13.8 REST Security

In addition to providing the ability to secure OWS style services, GeoServer also supports securing RESTful services.

As with layer and service security, RESTful security configuration is based on `security_roles`. The mapping of request URI to role is defined in a file named `rest.properties`, located in the `security` directory of the GeoServer data directory.

13.8.1 Syntax

The following syntax defines access control rules for RESTful services (parameters in brackets [] are optional):

```
uriPattern;method[,method,...]=role[,role,...]
```

The parameters are:

- **uriPattern**—*ant pattern* that matches a set of request URIs

- **method**—HTTP request method, one of `GET`, `POST`, `PUT`, `POST`, `DELETE`, or `HEAD`
- **role**—Name of a predefined role. The wildcard '* is used to indicate the permission is applied to all users, including anonymous users.

Note:

- URI patterns should account for the first component of the rest path, usually `rest` or `api`
- Method and role lists should **not** contain any spaces

Ant patterns

Ant patterns are commonly used for pattern matching directory and file paths. The *examples* section contains some basic instructions. The Apache ant user manual contains more sophisticated use cases.

13.8.2 Examples

Most of the examples in this section are specific to the GeoServer *REST* but any RESTful GeoServer service may be configured in the same manner.

Allowing only authenticated access

The most secure configuration is one that forces any request to be authenticated. The following example locks down access to all requests:

```
/**;GET,POST,PUT,DELETE=ROLE_ADMINISTRATOR
```

A less restricting configuration locks down access to operations under the path `/rest`, but will allow anonymous access to requests that fall under other paths (for example `/api`):

```
/rest/**;GET,POST,PUT,DELETE=ROLE_ADMINISTRATOR
```

The following configuration is similar to the previous one except it grants access to a specific role rather than the administrator:

```
/**;GET,POST,PUT,DELETE=ROLE_TRUSTED
```

`ROLE_TRUSTED` is a role defined in `users.properties`.

Providing anonymous read-only access

The following configuration allows anonymous access when the `GET` (read) method is used but forces authentication for a `POST`, `PUT`, or `DELETE` (write):

```
/**;GET=IS_AUTHENTICATED_ANONYMOUSLY
/**;POST,PUT,DELETE=TRUSTED_ROLE
```

Securing a specific resource

The following configuration forces authentication for access to a particular resource (in this case a feature type):

```
/rest/**/states*;GET=TRUSTED_ROLE
/rest/**;POST,PUT,DELETE=TRUSTED_ROLE
```

The following secures access to a set of resources (in this case all data stores):

```
/rest/**/datastores/*;GET=TRUSTED_ROLE
/rest/**/datastores/*.*;GET=TRUSTED_ROLE
/rest/**;POST,PUT,DELETE=TRUSTED_ROLE
```

13.9 Disabling security

If you are using an external security subsystem, you may want to disable the built-in security to prevent conflicts. Disabling security is possible for each security filter chain individually. The security filter chains are listed on the GeoServer authentication page.

Warning: Disabling security for a filter chain results in administrator privileges for each HTTP request matching this chain. As an example, disabling security on the **web** chain gives administrative access to each user accessing the *Web administration interface* interface.

13.10 Tutorials

13.10.1 Authentication with LDAP

This tutorial introduces GeoServer LDAP support and walks through the process of setting up authentication aganist an LDAP server. It is recommended that the *LDAP authentication* section be read before proceeding.

LDAP server setup

A mock LDAP server will be used for this tutorial. Download and run the acme-ldap jar:

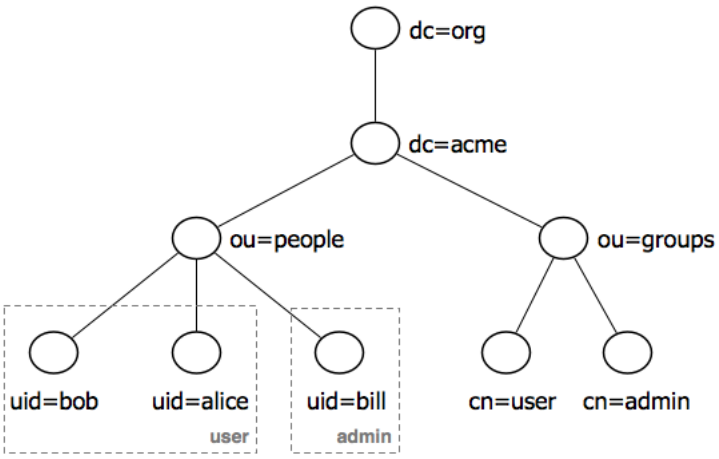
```
java -jar acme-ldap.jar
```

The output of which should look like the following:

```
Directory contents:
  ou=people,dc=acme,dc=org
    uid=bob,ou=people,dc=acme,dc=org
    uid=alice,ou=people,dc=acme,dc=org
    uid=bill,ou=people,dc=acme,dc=org
  ou=groups,dc=acme,dc=org
  cn=users,ou=groups,dc=acme,dc=org
    member: uid=bob,ou=people,dc=acme,dc=org
    member: uid=alice,ou=people,dc=acme,dc=org
```

```
cn=admins,ou=groups,dc=acme,dc=org
  member: uid=bill,ou=people,dc=acme,dc=org

Server running on port 10389
```

The following diagram illustrates the hierachy of the LDAP datatabse:

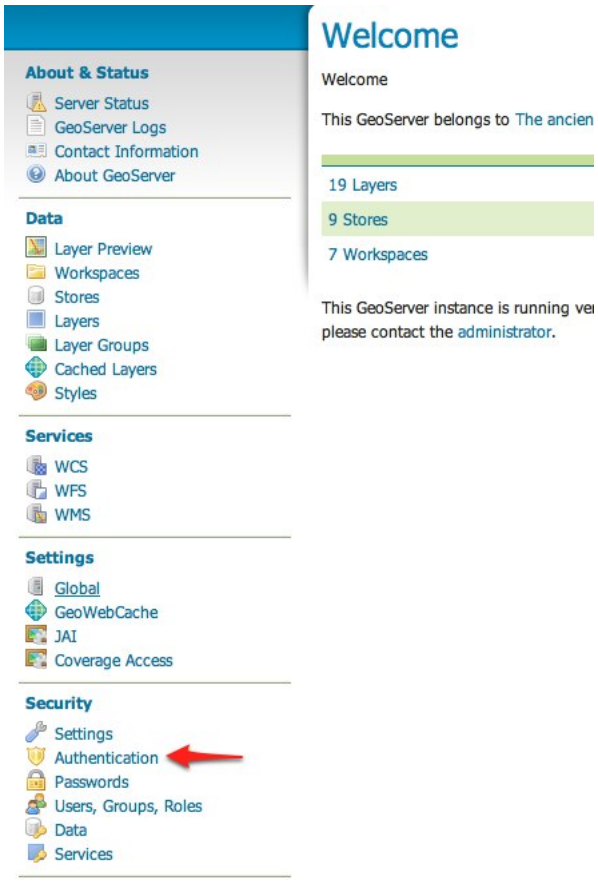


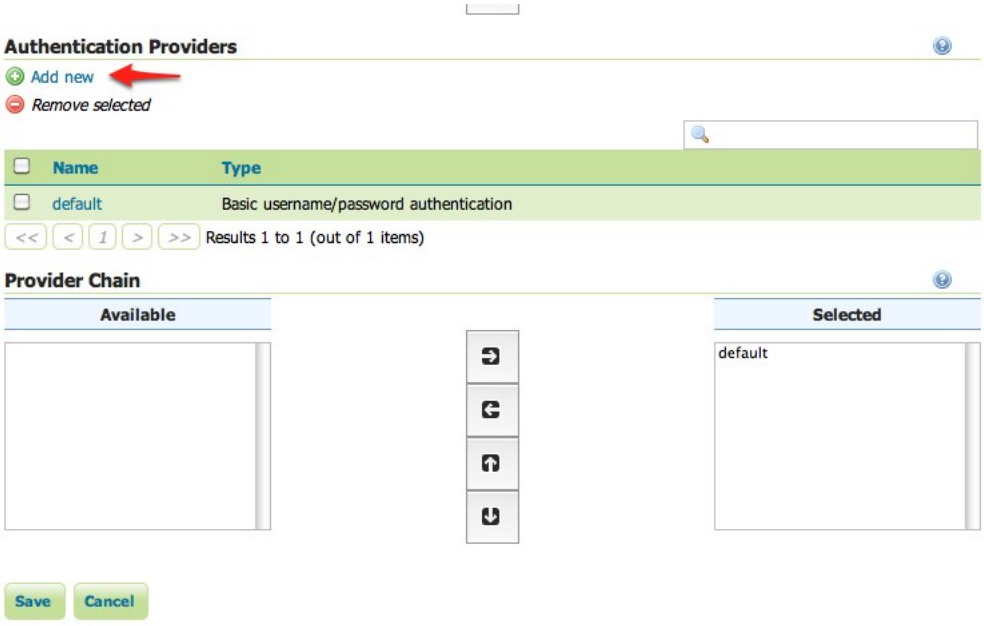
The LDAP tree consists of:

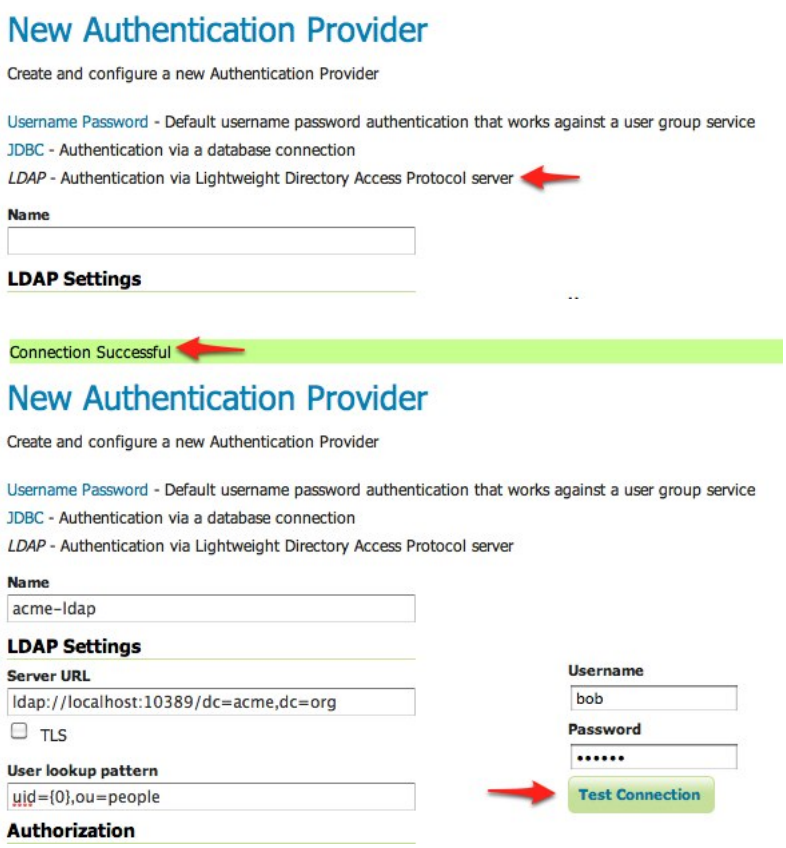
- The root domain component, `dc=acme,dc=org`
- Two organizational units (groups) named `user` and `admin`
- Two users named `bob` and `alice` who are members of the `user` group
- One user named `bill` who is a member of the `admin` group

Configure the LDAP authentication provider

1. Start GeoServer and login to the web admin interface as the `admin` user.
2. Click the `Authentication` link located under the `Security` section of the navigation sidebar.
3. Scroll down to the `Authentication Providers` panel and click the `Add new` link.
4. Click the `LDAP` link.
5. Fill in the fields of the settings form as follows:
   - Set `Name` to "acme-ldap"
   - Set `Server URL` to "ldap://localhost:10389/dc=acme,dc=org"
   - Set `User lookup pattern` to "uid={0},ou=people"


Welcome
About & Status
Server Status
GeoServer Logs
Contact Information
About GeoServer
Data
Layer Preview
Workspaces
Stores
Layers
Layer Groups
Cached Layers
Styles
Services
WCS
WFS
WMS
Settings
Global
GeoWebCache
JAI
Coverage Access
Security
Settings
Authentication
Passwords
Users, Groups, Roles
Data
Services
Welcome
This GeoServer belongs to The ancien
19 Layers
9 Stores
7 Workspaces
This GeoServer instance is running ve
please contact the administrator.


Authentication Providers
Add new
Remove selected
Name
Type
default
Basic username/password authentication
<< < 1 > >> Results 1 to 1 (out of 1 items)
Provider Chain
Available
Selected
default
Save
Cancel

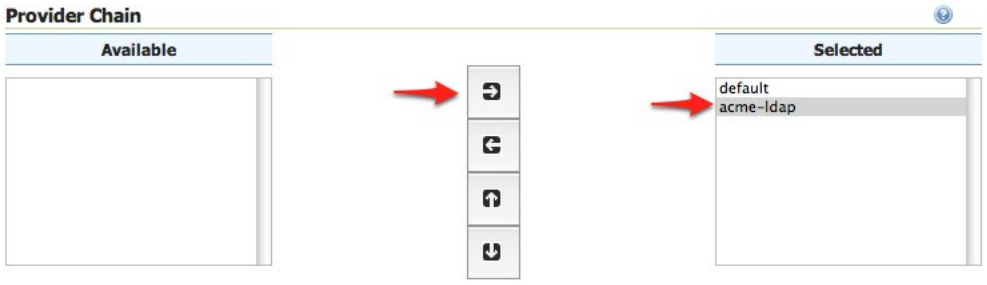


6. Test the LDAP connection by entering the username "bob" and password "secret" in the connection test form located on the right and click the `Test Connection` button.

   A successful connection should be reported at the top of the page.

7. Save.

8. Back on the authentication page scroll down to the `Provider Chain` panel and move the `acme-ldap` provider from `Available` to `Selected`.

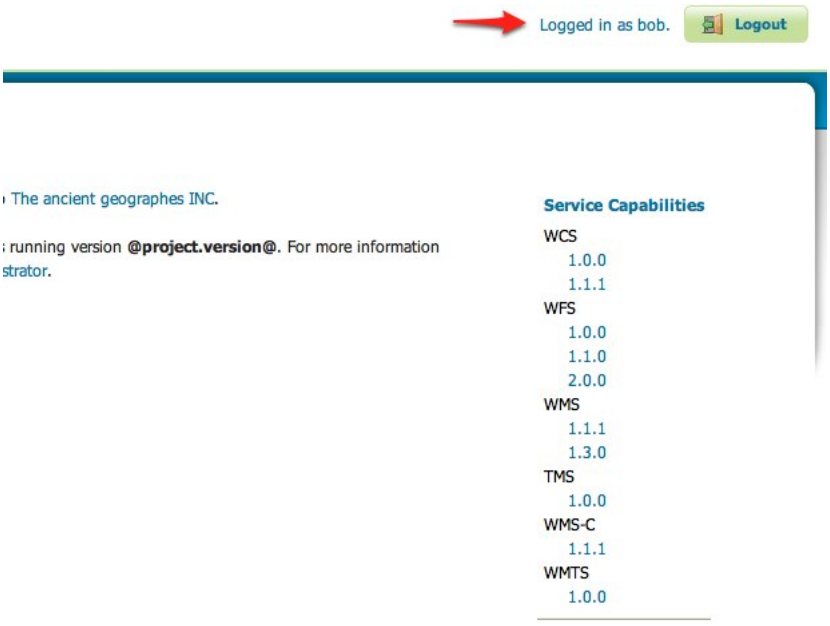


9. Save.

Test a LDAP login

1. Navigate to the GeoServer home page and log out of the admin account.

2. Login as the user “bob” with the with the password “secret”.

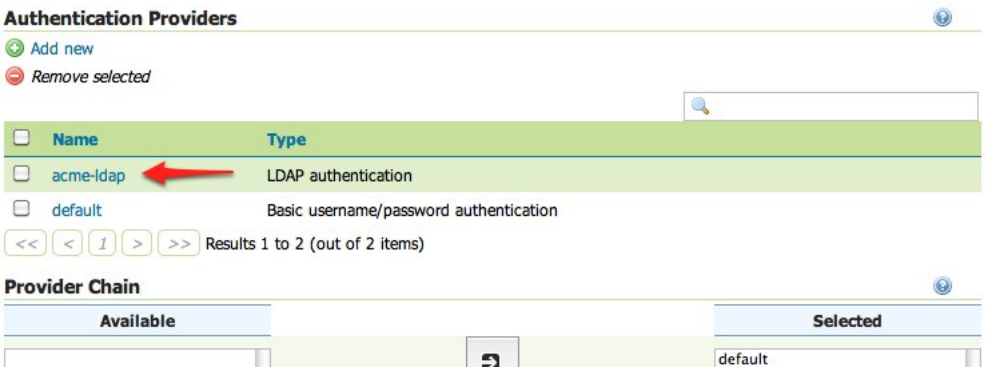


Logging in as bob doesn’t yield any administrative functionality because the bobaccount has not been mapped to the administrator role. In the next section GeoServer will be configured to map groups from the LDAP database to roles.

Map LDAP groups to GeoServer roles

When using LDAP for authentication GeoServer maps LDAP groups to GeoServer roles by prefixing the group name with `ROLE_` and converting the result to uppercase. For example bob and alice are members of the `user` group so after authentication they would be assigned a role named `ROLE_USER`. Similarily bill is a member of the `admin` group so he would be assigned a role named `ROLE_ADMIN`.

1. Log out of the web admin and log back in as the admin user.
2. Navigate to the `Authentication` page.
3. Scroll to the `Authentication Providers` panel and click the `acme-ldap` link.



4. On the settings page fill in the following form fields:

- Set `Group search base` to "ou=groups"
- Set `Group search filter` to "member={0}"

The first field specifies the node of the LDAP directory tree at which groups are located. In this case the organizational unit named `groups`. The second field specifies the LDAP query filter to use in order to locate those groups that a specific user is a member of. The `{0}` is a placeholder which is replaced with the `uid` of the user.

- Set `Group to use as ADMIN` to "ADMIN"
- Set `Group to use as GROUP_ADMIN` to "ADMIN"

These settings let users in the LDAP admin group to be recognized as GeoServer administrators.

5. Save.

At this point the LDAP provider will populate an authenticated user with roles based on the groups the user is a member of.

At this point members of the `admin` LDAP group should be given full administrative privileges once authenticated. Log out of the admin account and log in as "bill" with the password "hello". Once logged in full administrative functionality should be available.

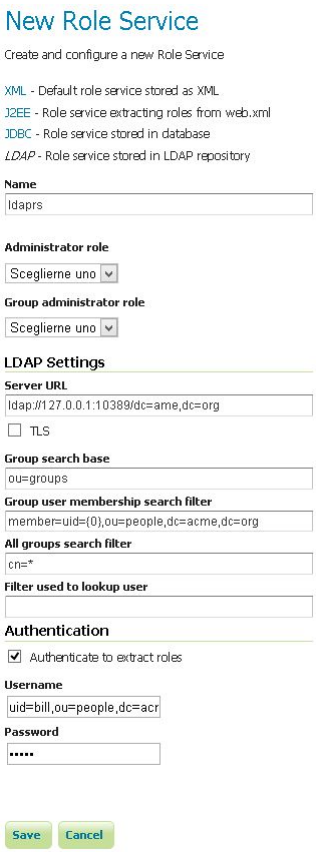
Configure the LDAP role service

An additional step permits to configure a role service to get GeoServer roles from the LDAP repository and allow access rights to be assigned to those roles.

1. Click the `Users,Group,Roles` link located under the `Security` section of the navigation sidebar.
2. Click the `Add new link` under the `Role Services` section.
3. Click the `LDAP` option under the `New Role Service` section.
4. Enter `ldaprs` in the `Name` text field.
5. Enter `ldap://localhost:10389/dc=acme,dc=org` in the `Server URL` text field.
6. Enter `ou=groups` in the `Group search base` text field.
7. Enter `member=uid={0},ou=people,dc=acme,dc=org` in the `Group user membership search filter` text field.
8. Enter `cn=*` in the `All groups search filter` text field.

Then we need to a choose a user to authenticate on the server (many LDAP server don't allow anonymous data lookup).

1. Check the `Authenticate to extract roles` checkbox.
2. Enter `uid=bill,ou=people,dc=acme,dc=org` in the `Username` text field.
3. Enter `hello` in the `Password` text field.
4. Save.
5. Click the `ldaprs` role service item under the `Role Services` section.
6. Select `ROLE_ADMIN` from the `Administrator role` combobox.
7. Select `ROLE_ADMIN` from the `Group administrator role` combobox.
8. Save again.



You should now be able to see and assign the new `ROLE_ADMIN` and `ROLE_USER` roles wherever an `Available Roles` list is shown (for example in the `Data` and `Services` rules sections.

13.10.2 Authentication with LDAP against ActiveDirectory

This tutorial explains how to use GeoServer LDAP support to connect to a Windows Domain using ActiveDirectory as an LDAP server. It is recommended that the *LDAP authentication* section be read before proceeding.

Windows Server and ActiveDirectory

Active Directory is just another LDAP server implementation, but has some features that we must know to successfully use it with GeoServer LDAP authentication. In this tutorial we will assume to have a Windows Server Domain Controller with ActiveDirectory named `domain-controller` for a domain named `ad.local`. If your environment uses different names (and it surely will) use your real names where needed.

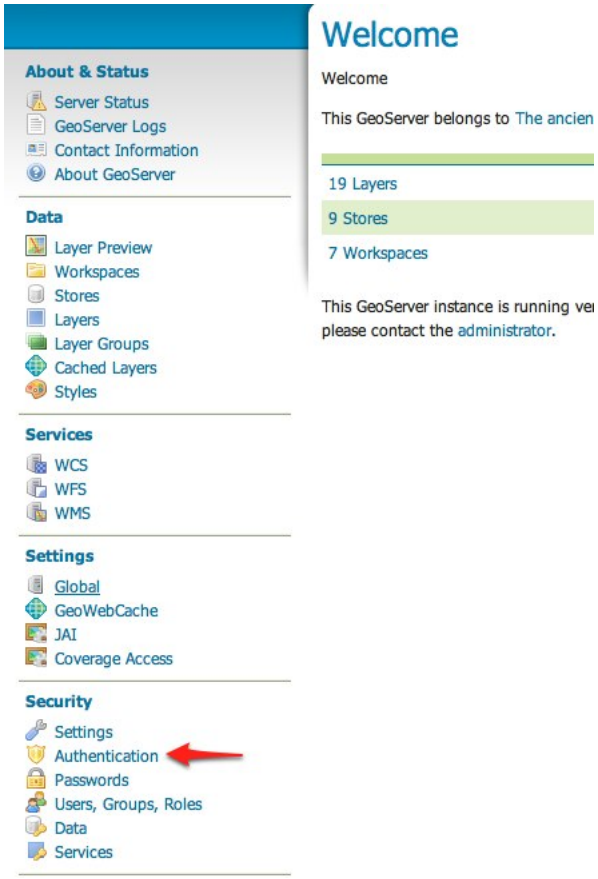
We will also assume that:

- a group named `GISADMINGROUP` exists.
- a user named `GISADMIN` exists, has password `secret`, and belongs to the `GISADMINGROUP` group.
- a user named `GISUSER` exists, has password `secret`, and does NOT belong to the `GISADMINGROUP` group.

Note: ADMINISTRATOR cannot be generally used as the admin group name with ActiveDirectory, because Administrator is the master user name in Windows environment.

Configure the LDAP authentication provider

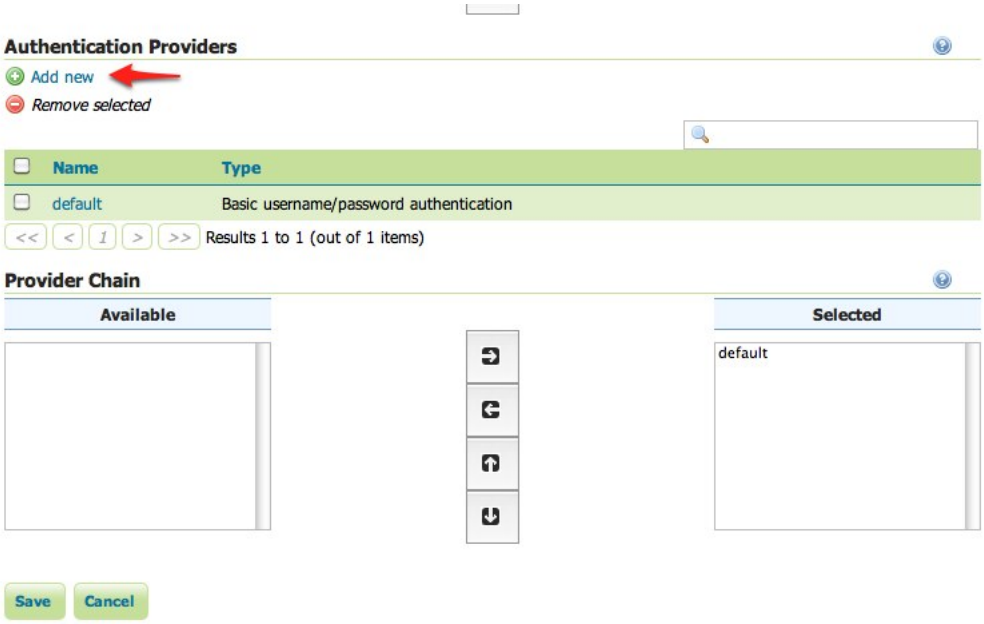
1. Start GeoServer and login to the web admin interface as the `admin` user.
2. Click the `Authentication` link located under the `Security` section of the navigation sidebar.

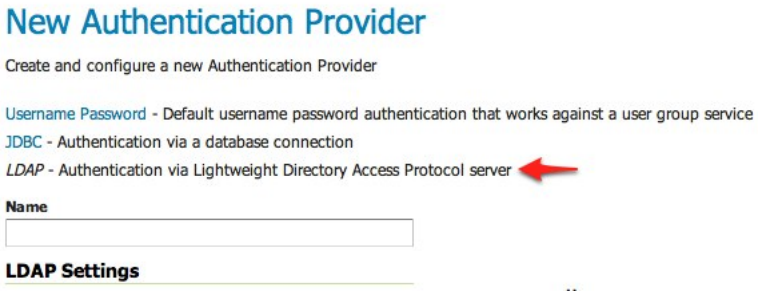


3. Scroll down to the `Authentication Providers` panel and click the `Add new` link.

4. Click the `LDAP` link.

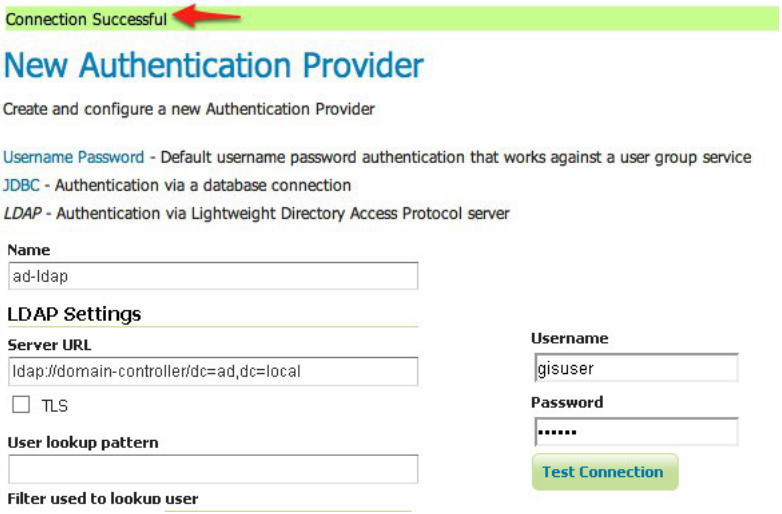
5. Fill in the fields of the settings form as follows:
   - Set `Name` to "ad-ldap"
   - Set `Server URL` to "ldap://domain-controller/dc=ad,dc=local
   - Set `Filter used to lookup user` to `(|(userPrincipalName={0})(sAMAccountName={1}))`
   - Set `Format used for user login name` to "{0}@ad.local"


Authentication Providers
Add new
Remove selected
Name
Type
default
Basic username/password authentication
<< < 1 > >> Results 1 to 1 (out of 1 items)
Provider Chain
Available
Selected
default
Save
Cancel


New Authentication Provider
Create and configure a new Authentication Provider
Username Password - Default username password authentication that works against a user group service
JDBC - Authentication via a database connection
LDAP - Authentication via Lightweight Directory Access Protocol server
Name
LDAP Settings

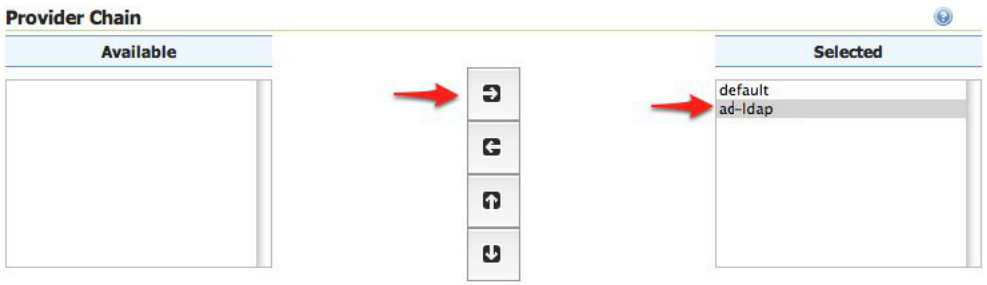
- Check `Use LDAP groups for authorization`
- Check `Bind user before searching for groups`
- Set `Group to use as ADMIN` to "GISADMINGROUP"
- Set `Group search base` to "cn=Users"
- Set `Group search filter` to "member={0}"

6. Test the LDAP connection by entering the username "GISADMIN" and password "secret" in the connection test form located on the right and click the `Test Connection` button.



A successful connection should be reported at the top of the page.

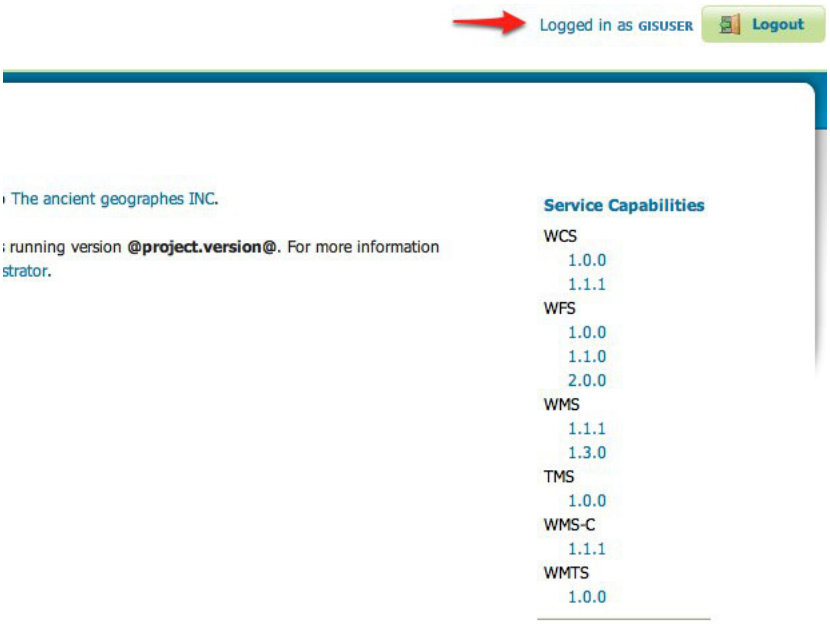
7. Save.
8. Back on the authentication page scroll down to the `Provider Chain` panel and move the `ad-ldap` provider from `Available` to `Selected`.



9. Save.

Test a LDAP login

1. Navigate to the GeoServer home page and log out of the admin account.
2. Login as the user "GISUSER" with the with the password "secret".



Logging in as GISUSER doesn't yield any administrative functionality because the GISUSER account has not been mapped to the administrator role. In the next section GeoServer will be configured to map groups from the LDAP database to roles.

Now we will login with a user having administrative rights.

1. Navigate to the GeoServer home page and log out of the account.
2. Login as the user "GISADMIN" with the with the password "secret".

Once logged in full administrative functionality should be available.

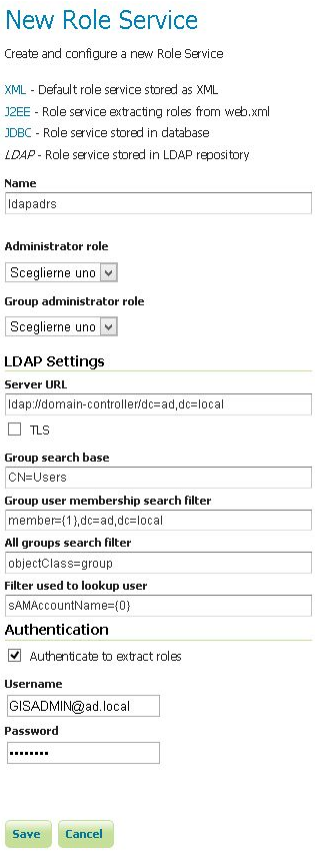
Configure the LDAP role service

An additional step permits to configure a role service to get GeoServer roles from the LDAP repository and allow access rights to be assigned to those roles.

1. Click the `Users,Group,Roles` link located under the `Security` section of the navigation sidebar.
2. Click the `Add new link` under the `Role Services` section.
3. Click the `LDAP` option under the `New Role Service` section.
4. Enter `ldapadrs` in the `Name` text field.
5. Enter `ldap://domain-controller/dc=ad,dc=local` in the `Server URL` text field.
6. Enter `CN=Users` in the `Group search base` text field.
7. Enter `member={1},dc=ad,dc=local` in the `Group user membership search filter` text field.
8. Enter `objectClass=group` in the `All groups search filter` text field.
9. Enter `sAMAccountName={0}` in the `Filter used to lookup user` text field.

Then we need to a choose a user to authenticate on the server (many LDAP server don't allow anonymous data lookup).

1. Check the `Authenticate to extract roles` checkbox.



2. Enter `GISADMIN@ad.local` in the `Username` text field.
3. Enter `secret` in the `Password` text field.
4. Save.
5. Click the `ldapadrs` role service item under the `Role Services` section.
6. Select `ROLE_DOMAIN ADMINS` from the `Administrator role` combobox.
7. Select `ROLE_DOMAIN ADMINS` from the `Group administrator role` combobox.
8. Save again.

You should now be able to see and assign the new ActiveDirectory roles wherever an `Available Roles` list is shown (for example in the `Data` and `Services` rules sections.

13.10.3 Configuring Digest Authentication

Introduction

Out of the box GeoServer REST and OGC services support authentication via HTTP Basic authentication. One of the major downsides of basic auth is that it sends user passwords in plain text. HTTP Digest authentication offers a more secure alternative that applies a cryptographic hash function to passwords before sending them over the network.

This tutorial walks through the process of setting up digest authentication.

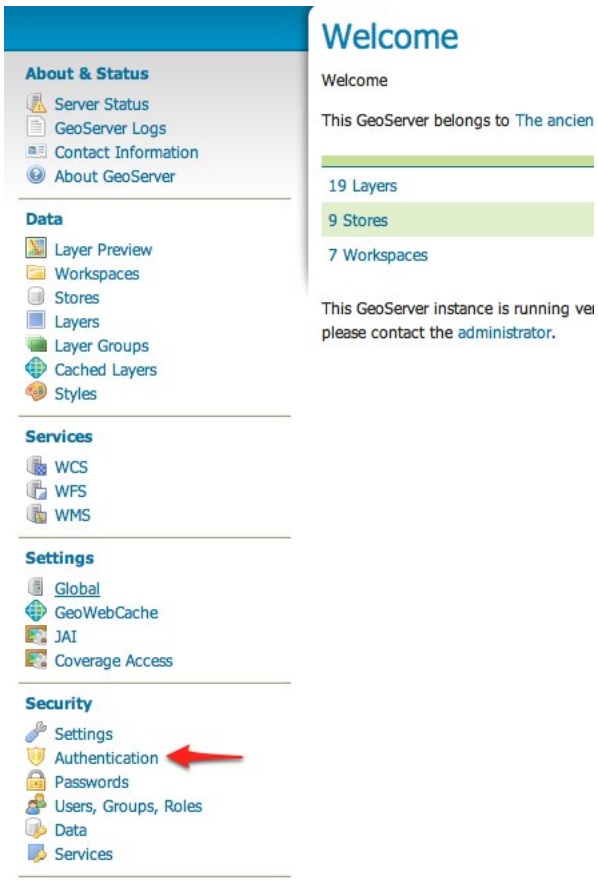
Prerequisites

This tutorial uses the curl utility to issue HTTP request that test authentication. Install curl before proceeding.

Note: Any utility that supports both basic and digest authentication can be used in place of curl. Most modern web browsers support both types of authentication.

Configure the Digest authentication filter

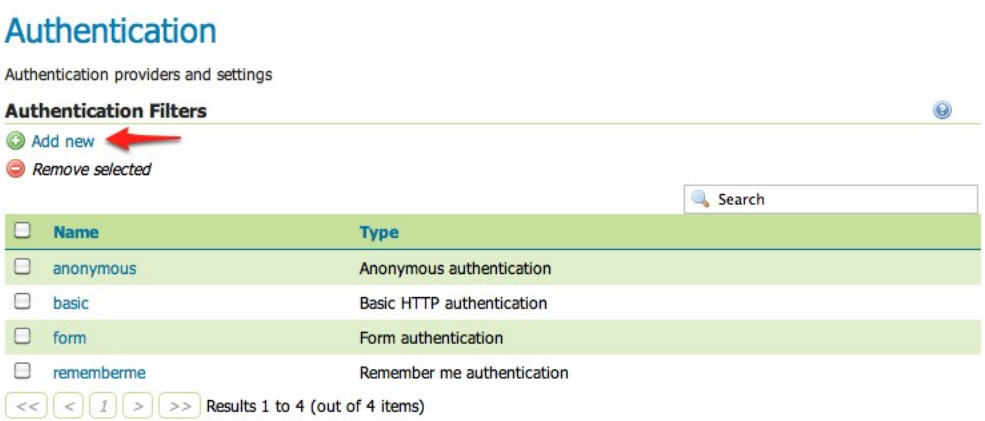
1. Start GeoServer and login to the web admin interface as the `admin` user.
2. Click the `Authentication` link located under the `Security` section of the navigation sidebar.



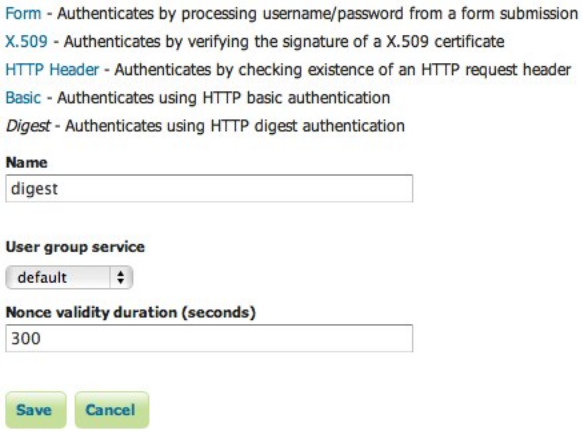
3. Scroll down to the `Authentication Filters` panel and click the `Add new` link.

4. Click the `Digest` link.

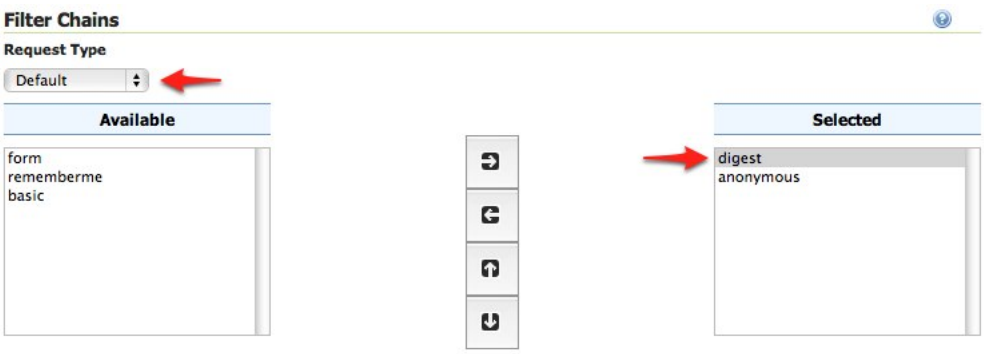
5. Fill in the fields of the settings form as follows:
   - Set `Name` to "digest"



- Set `User group service` to "default"



6. Save.
7. Back on the authentication page scroll down to the `Filter Chains` panel.
8. Select "Default" from the `Request type` drop down.
9. Unselect the `basic` filter and select the `digest` filter. Position the the `digest` filter before the `anonymous` filter.
10. Save.



Secure OGC service requests

In order to test the authentication settings configured in the previous section a service or resource must be first secured. The `Default` filter chain is the chain applied to all OGC service requests so a service security rule must be configured.

1. From the GeoServer home page and click the `Services` link located under the `Security` section of the navigation sidebar.

2. On the Service security page click the `Add new rule` link and add a catch all rule that secures all OGC service requests requiring the `ROLE_ADMINISTRATOR` role.
3. Save.

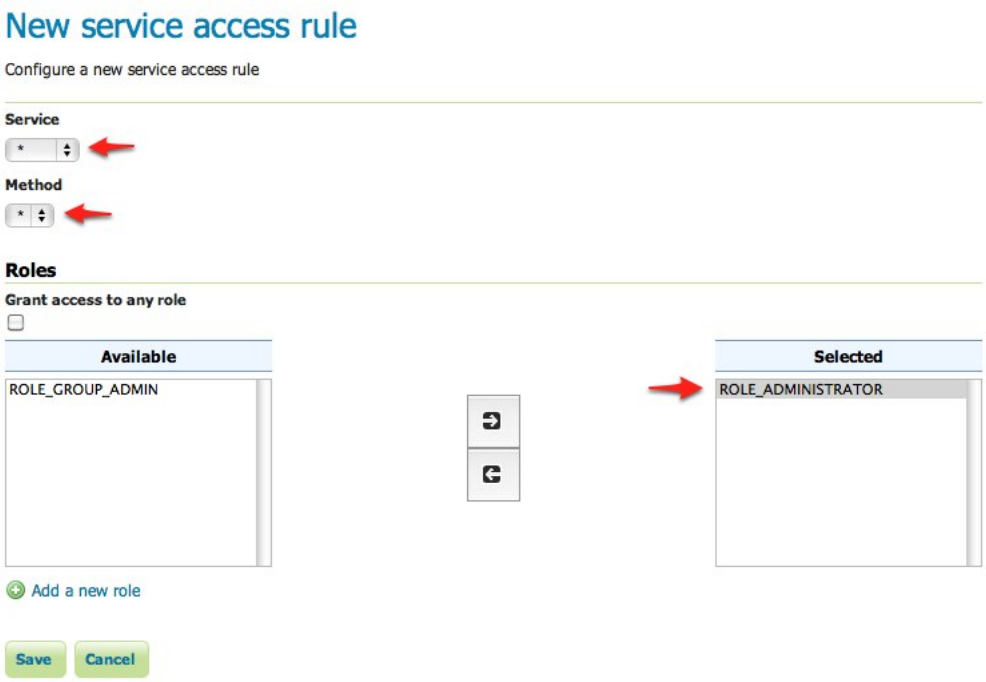
Test a digest authentication login

1. Ensure that basic authentication is disabled execute the following curl command:

```
curl -v -u admin:geoserver -G "http://localhost:8080/geoserve/wfs?
↪request=getcapabilities"
```

The result should be a 401 response signaling that authentication is required. The output should look something like the following:

```
* About to connect() to localhost port 8080 (#0)
*   Trying 127.0.0.1... connected
* Connected to localhost (127.0.0.1) port 8080 (#0)
* Server auth using Basic with user 'admin'
> GET /geoserver/wfs?request=getcapabilities HTTP/1.1
> Authorization: Basic YWRtaW46Z2Vvc2VydmVy
> User-Agent: curl/7.19.7 (universal-apple-darwin10.0) libcurl/7.19.7 OpenSSL/0.9.
↪8r zlib/1.2.3
> Host: localhost:8080
> Accept: */*
>
```




```
< HTTP/1.1 401 Full authentication is required to access this resource
< Set-Cookie: JSESSIONID=1dn2bi8qqu5qc;Path=/geoserver
< WWW-Authenticate: Digest realm="GeoServer Realm", qop="auth", nonce=
↪"MTMzMzQzMDkxMTU3MjphZGIwMWE4MTc1NmRiMzI3YmFiODhmY2NmZGQ2MzEwZg=="
< Content-Type: text/html; charset=iso-8859-1
< Content-Length: 1491
< Server: Jetty(6.1.8)
<
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1"/>
<title>Error 401 Full authentication is required to access this resource</title>
</head>
...
```

2. Execute the same command but specify the `--digest` option to tell curl to use digest authentication rather than basic authentication:

```
curl --digest -v -u admin:geoserver -G "http://localhost:8080/geoserve/wfs?
↪request=getcapabilities"
```

The result should be a successful authentication and contain the normal WFS capabilities response.

13.10.4 Configuring X.509 Certificate Authentication

Certificate authentication involves the usage of public/private keys to identify oneself. This represents a much more secure alternative to basic user name and password schemes.

X.509 is a well defined standard for the format of public key certificates. This tutorial walks through the process of setting up X.509 certificate authentication.

Prerequisites

This tutorial assumes the following:

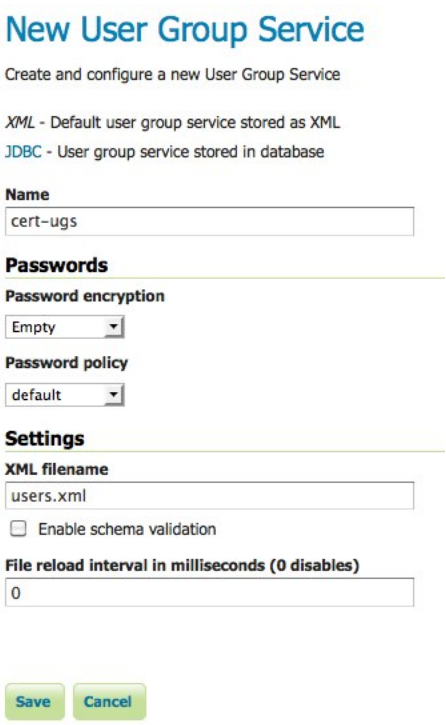
- A web browser that supports the usage of client certificates for authentication, also referred to as "two-way SSL". This tutorial uses **Firefox**.
- An SSL-capable servlet container. This tutorial uses **Tomcat**.
- GeoServer is deployed in Tomcat.

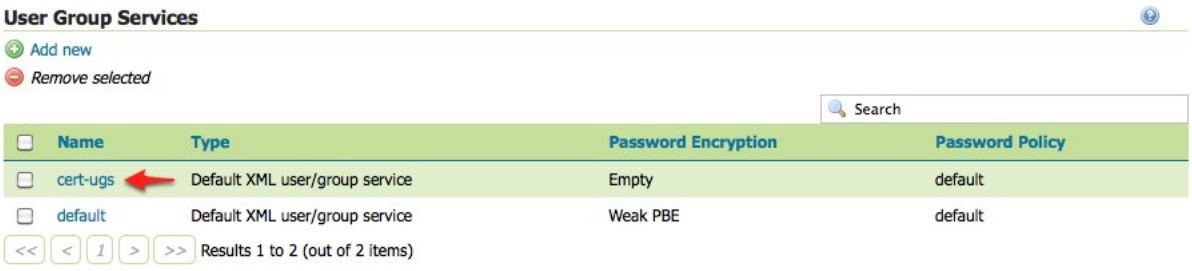
Configure the user/group service

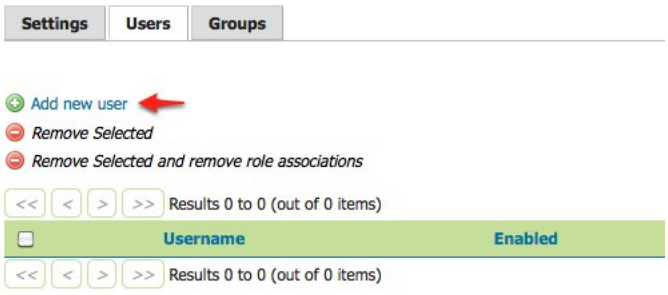
Users authenticated via a X.509 certificate must be configured in GeoServer. For this a new user/group service will be added.

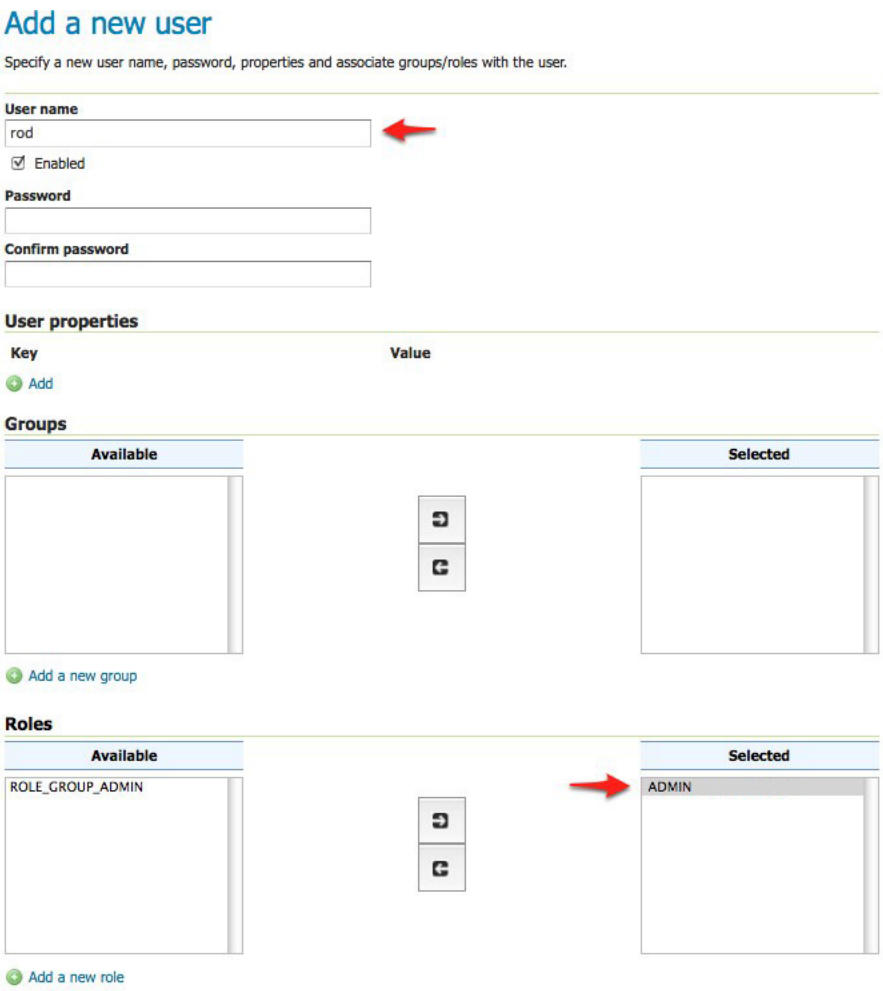
1. Login to the web admin interface as the `admin` user.
2. Click the `Users, Groups, and Roles` link located under the `Security` section of the navigation sidebar.

3. Scroll down to the `User Group Services` panel and click the `Add new` link.
4. Create a new user/group service named `cert-ugs` and fill out the settings form as follows:
   - Set *Password encryption* to `Empty` since users will not authenticate via password.
   - Set *Password policy* to `default`.
5. Click *Save*.
6. Back on the `Users, Groups, and Roles` page, click the *cert-ugs* link.
7. Select the *Users* tab and click the *Add new user* link.
8. Add a new user named `rod` the and assign the `ADMIN` role.
9. Click *Save*.
10. Click the *Authentication* link located under the *Security* section of the navigation sidebar.
11. Scroll down to the *Authentication Filters* panel and click the *Add new* link.
12. Click the *X.509* link and fill out form as follows:
    - Set *Name* to "cert"
    - Set *Role source* to `User group service` and set the associated drop-down to `cert-ugs`
13. Click *Save*.
14. Back on the authentication page, scroll down to the *Filter Chains* panel.

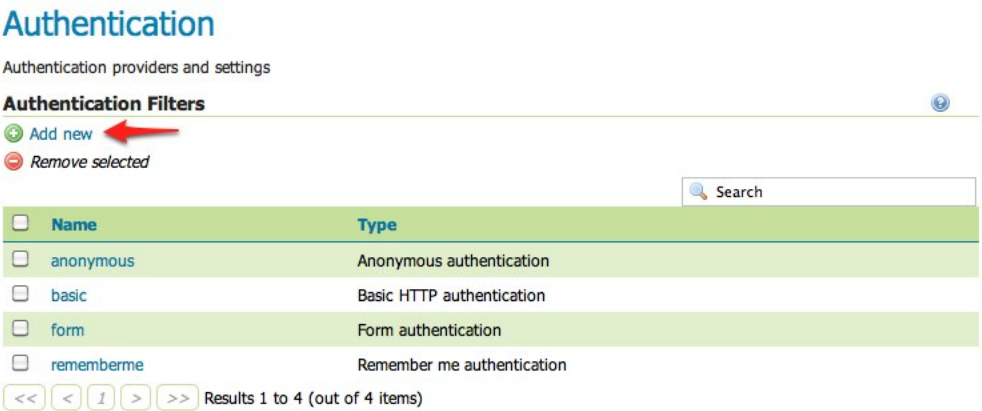

New User Group Service
Create and configure a new User Group Service
XML - Default user group service stored as XML
JDBC - User group service stored in database
Name
cert-ugs
Passwords
Password encryption
Empty
Password policy
default
Settings
XML filename
users.xml
Enable schema validation
File reload interval in milliseconds (0 disables)
0
Save
Cancel

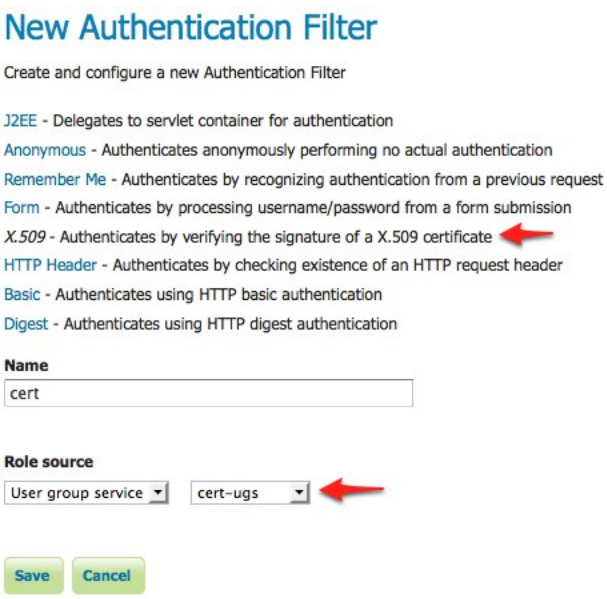

User Group Services
Add new
Remove selected
Search
Name
Type
Password Encryption
Password Policy
cert-ugs
Default XML user/group service
Empty
default
default
Default XML user/group service
Weak PBE
default
<< < 1 > >> Results 1 to 2 (out of 2 items)


Settings
Users
Groups
Add new user
Remove Selected
Remove Selected and remove role associations
<< < > >> Results 0 to 0 (out of 0 items)
Username
Enabled
<< < > >> Results 0 to 0 (out of 0 items)

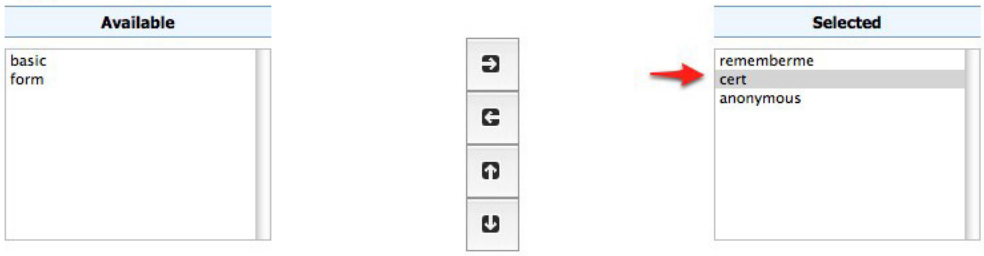

Add a new user
Specify a new user name, password, properties and associate groups/roles with the user.
User name
rod
Enabled
Password
Confirm password
User properties
Key
Value
Add
Groups
Available
Selected
Add a new group
Roles
Available
Selected
ROLE_GROUP_ADMIN
ADMIN
Add a new role


Security
Settings
Authentication
Passwords
Users, Groups, Roles
Data
Services


Authentication
Authentication providers and settings
Authentication Filters
Add new
Remove selected
Search
Name
Type
anonymous
Anonymous authentication
basic
Basic HTTP authentication
form
Form authentication
rememberme
Remember me authentication
<< < 1 > >> Results 1 to 4 (out of 4 items)



15. Click *web* in the *Name* column.
16. Select the *cert* filter and position it after the *rememberme* filter.



17. Click *Close*.
18. You will be returned to the previous page. Click *Save*.

> **Warning:** This last change requires both *Close* and then *Save* to be clicked. You may wish to return to the *web* dialog to verify that the change was made.

Download sample certificate files

Rather than demonstrate how to create or obtain valid certificates, which is beyond the scope of this tutorial, sample files available as part of the spring security sample applications will be used.

Download and unpack the `sample certificate files`. This archive contains the following files:

- `ca.pem` is the certificate authority (CA) certificate issued by the "Spring Security Test CA" certificate authority. This file is used to sign the server and client certificates.
- `server.jks` is the Java keystore containing the server certificate and private key used by Tomcat and presented to the user during the setup of the SSL connection.

- `rod.p12` contains the client certificate / key combination used to perform client authentication via the web browser.

Configure Tomcat for SSL

1. Copy the `server.jks` file into the `conf` directory under the root of the Tomcat installation.
2. Edit the Tomcat `conf/server.xml` and add an SSL connector:

```
<Connector port="8443" protocol="HTTP/1.1" SSLEnabled="true" scheme="https"␣
↪secure="true"
     clientAuth="true" sslProtocol="TLS"
     keystoreFile="${catalina.home}/conf/server.jks"
     keystoreType="JKS" keystorePass="password"
     truststoreFile="${catalina.home}/conf/server.jks"
     truststoreType="JKS" truststorePass="password" />
```

   This enables SSL on port 8443.

3. By default, Tomcat has APR enabled. To disable it so the above configuration can work, remove or comment out the following line in the server.xml configration file

```
<Listener className="org.apache.catalina.core.AprLifecycleListener"␣
↪SSLEngine="on" />
```

4. Restart Tomcat.

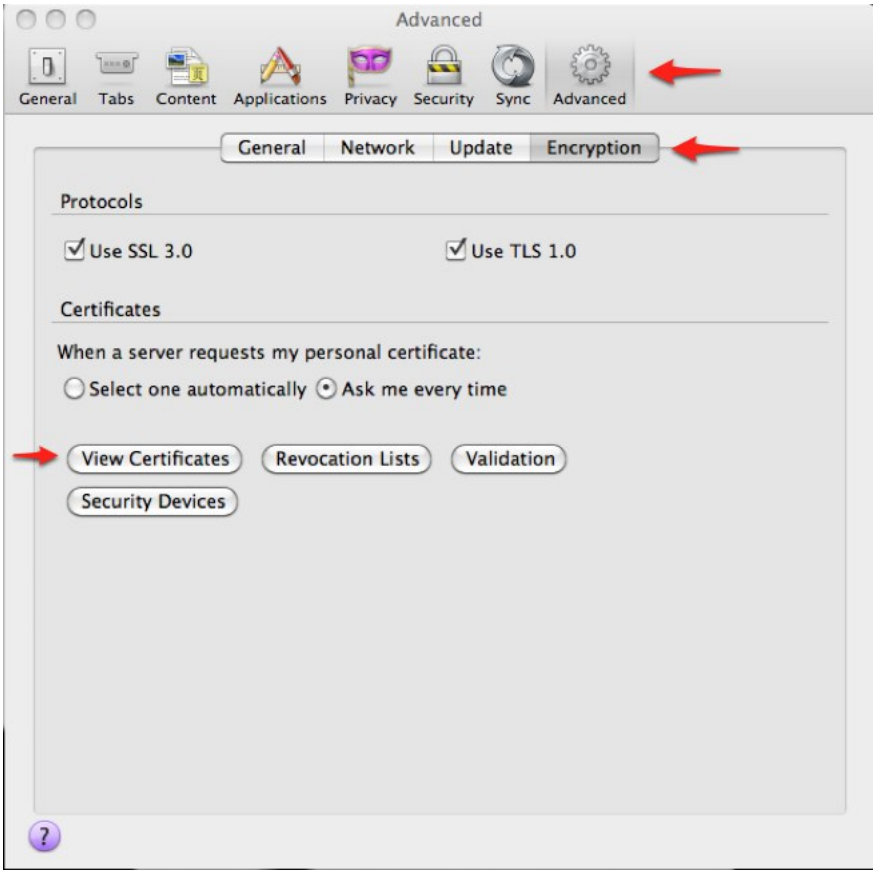
Install the client certificate

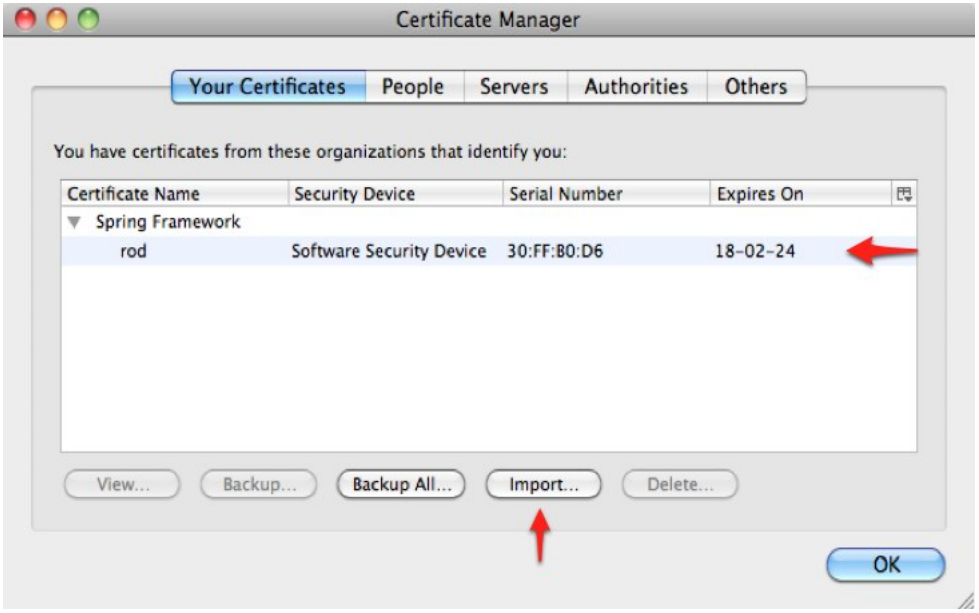
1. In Firefox, select *Preferences* (or *Tools* → *Options*) and navigate to the *Advanced* panel.
2. Select the *Encryption* tab (or the *Certificates* tab, depending on your version) and click the *View Certificates* button.
3. On the *Your Certificates* panel click the *Import* button and select the `rod.p12` file.
4. When prompted enter in the password `password`.
5. Click *OK* and close the Firefox Preferences.

Test certificate login

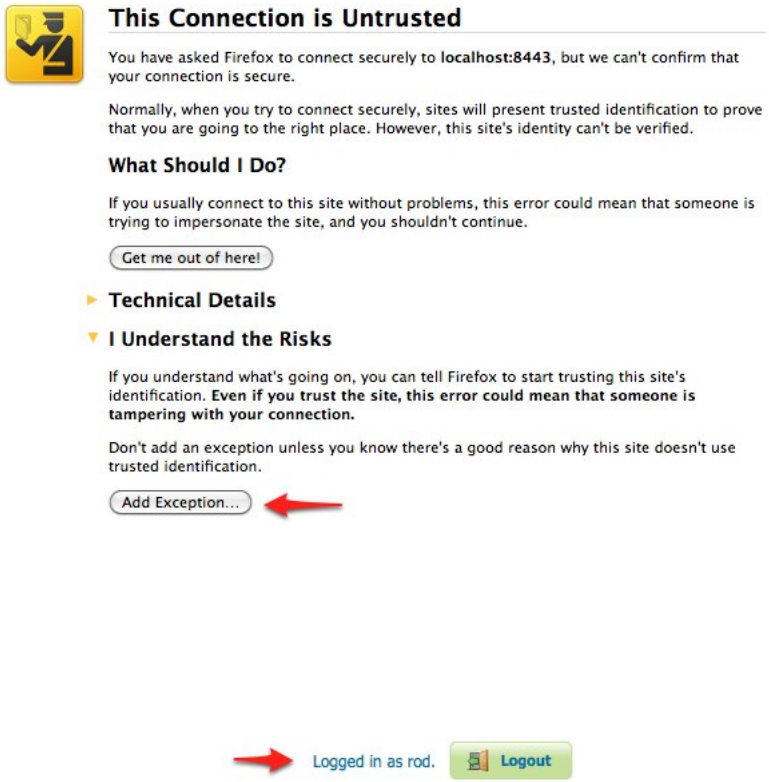
1. Navigate to the GeoServer admin on port "8443" using HTTPS: https://localhost:8443/geoserver/web
2. You will be prompted for a certificate. Select the *rod* certificate for identification.
3. When warned about the self-signed server certificate, click *Add Excception* to add a security exception.

The result is that the user `rod` is now logged into the GeoServer admin interface.


Advanced
General
Tabs
Content
Applications
Privacy
Security
Sync
Advanced
General
Network
Update
Encryption
Protocols
Use SSL 3.0
Use TLS 1.0
Certificates
When a server requests my personal certificate:
Select one automatically
Ask me every time
View Certificates
Revocation Lists
Validation
Security Devices

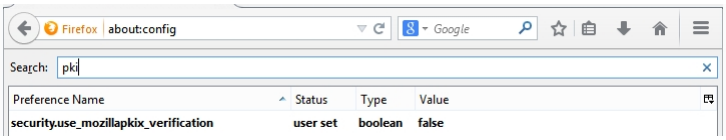

Certificate Manager
Your Certificates
People
Servers
Authorities
Others
You have certificates from these organizations that identify you:
Certificate Name
Security Device
Serial Number
Expires On
Spring Framework
rod
Software Security Device
30:FF:B0:D6
18-02-24
View...
Backup...
Backup All...
Import...
Delete...
OK


This site has requested that you identify yourself with a certificate:
localhost:8443
Organization: "Spring Security"
Issued Under: "Spring Framework"
Choose a certificate to present as identification:
rod [30:FF:B0:D6]
Details of selected certificate:
Issued to: CN=rod,OU=Spring Security,O=Spring Framework
Serial Number: 30:FF:B0:D6
Valid from 08-01-25 5:50:22 to 18-02-24 17:00:00
Certificate Key Usage: Signing
Issued by: CN=Spring Security Test CA,OU=Spring Security,O=Spring Framework,L=Glasgow,ST=Scotland,C=GB
Stored in: Software Security Device
Remember this decision
Cancel
OK


This Connection is Untrusted
You have asked Firefox to connect securely to localhost:8443, but we can't confirm that your connection is secure.
Normally, when you try to connect securely, sites will present trusted identification to prove that you are going to the right place. However, this site's identity can't be verified.
What Should I Do?
If you usually connect to this site without problems, this error could mean that someone is trying to impersonate the site, and you shouldn't continue.
Get me out of here!
Technical Details
I Understand the Risks
If you understand what's going on, you can tell Firefox to start trusting this site's identification. Even if you trust the site, this error could mean that someone is tampering with your connection.
Don't add an exception unless you know there's a good reason why this site doesn't use trusted identification.
Add Exception...

Logged in as rod.
Logout

Note: Starting with version 31, Firefox implements a new mechanism for using certificates, which will cause a *Issuer certificate is invalid error (sec_error_ca_cert_invalid)* error when trying to use a self-signed repository such as the one proposed. To avoid that, you can disable this mechanism by browsing to **about:config** and setting the **security.use_mozillapkix_verification** parameter to **false**.



13.10.5 Configuring J2EE Authentication

Servlet containers such as Tomcat and Jetty offer their own options for authentication. Often it is desirable for an application such as GeoServer to use that existing authentication mechanisms rather than require its own authentication configuration.

J2EE authentication allows GeoServer to delegate to the servlet container for authentication. This tutorial walks through the process of setting up J2EE authentication.

Prerequisites

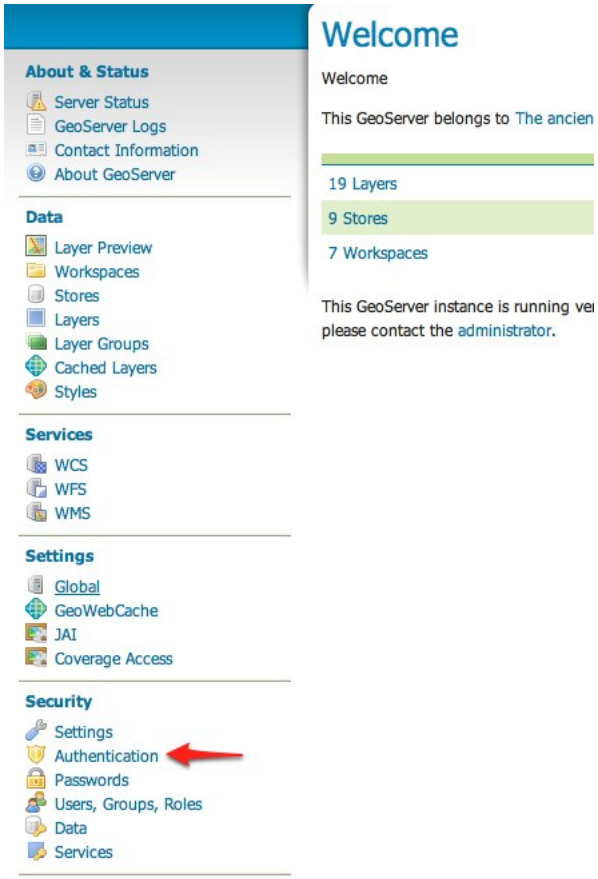
This tutorial requires a servlet container capable of doing its own authentication. This tutorial uses Tomcat.

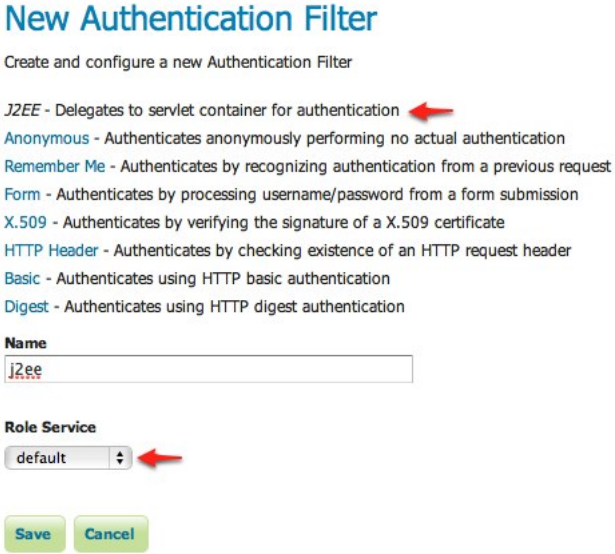
Deploy GeoServer in tomcat before proceeding.

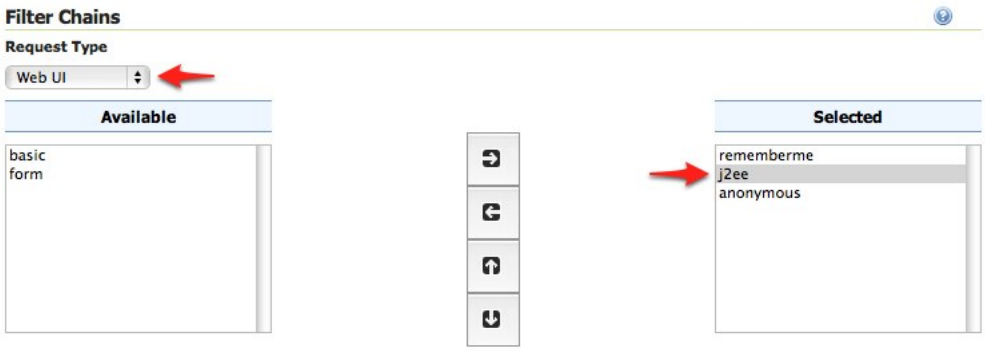
Configure the J2EE authentication filter

In order to delegate to the container for authentication a filter must first be configured to recognize the container authentication.

1. Login to the GeoServer web admin interface as the `admin` user.
2. Click the `Authentication` link located under the `Security` section of the navigation sidebar.
3. Scroll down to the `Authentication Filter` panel and click the `Add new` link.
4. Create a new filter named "j2ee" and fill out the settings form as follows:
   - Set the `Role service` to "default"
5. Save
6. Back on the authentication page scroll down to the `Filter Chains` panel.
7. Select "Web UI" from the `Request type` drop down.
8. Select the `j2ee` filter and position it after the `anonymous` filter.
9. Save.


Welcome
About & Status
Server Status
GeoServer Logs
Contact Information
About GeoServer
Data
Layer Preview
Workspaces
Stores
Layers
Layer Groups
Cached Layers
Styles
Services
WCS
WFS
WMS
Settings
Global
GeoWebCache
JAI
Coverage Access
Security
Settings
Authentication
Passwords
Users, Groups, Roles
Data
Services
Welcome
This GeoServer belongs to The ancien
19 Layers
9 Stores
7 Workspaces
This GeoServer instance is running ve
please contact the administrator.


New Authentication Filter
Create and configure a new Authentication Filter
J2EE - Delegates to servlet container for authentication
Anonymous - Authenticates anonymously performing no actual authentication
Remember Me - Authenticates by recognizing authentication from a previous request
Form - Authenticates by processing username/password from a form submission
X.509 - Authenticates by verifying the signature of a X.509 certificate
HTTP Header - Authenticates by checking existence of an HTTP request header
Basic - Authenticates using HTTP basic authentication
Digest - Authenticates using HTTP digest authentication
Name
j2ee
Role Service
default
Save
Cancel



Configure the role service

Since it is not possible to ask a J2EE container for the roles of a principal it is necessary to have all J2EE roles enlisted in a role service. The only J2EE API GeoServer can use is:

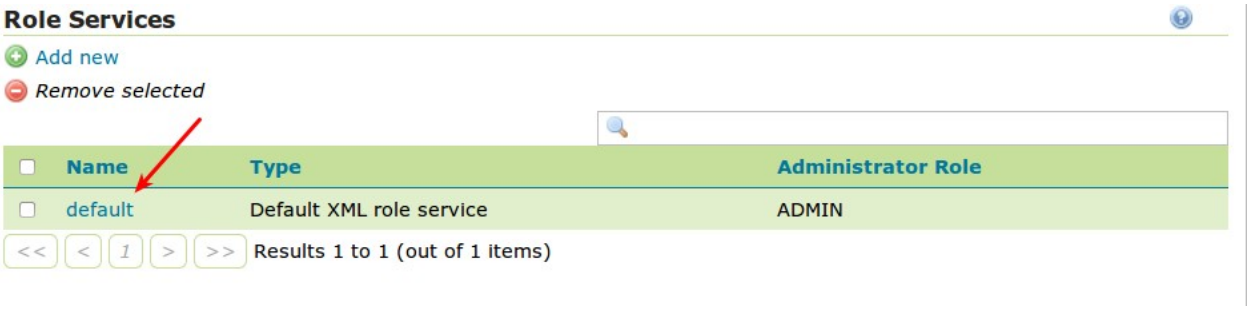
```
class: javax.servlet.http.HttpServletRequest
method: boolean isUserInRole(String role)
```

The idea is to query all roles from the role service and test each role with the "isUserInRole" method.

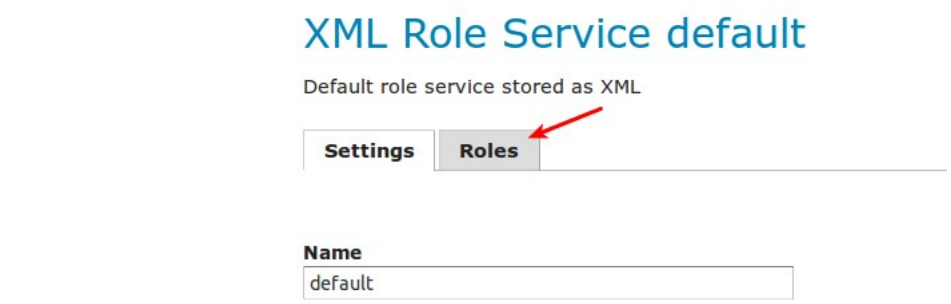
This tutorial assumes a user named "admin" with password "password" and a J2EE role named "tomcat".

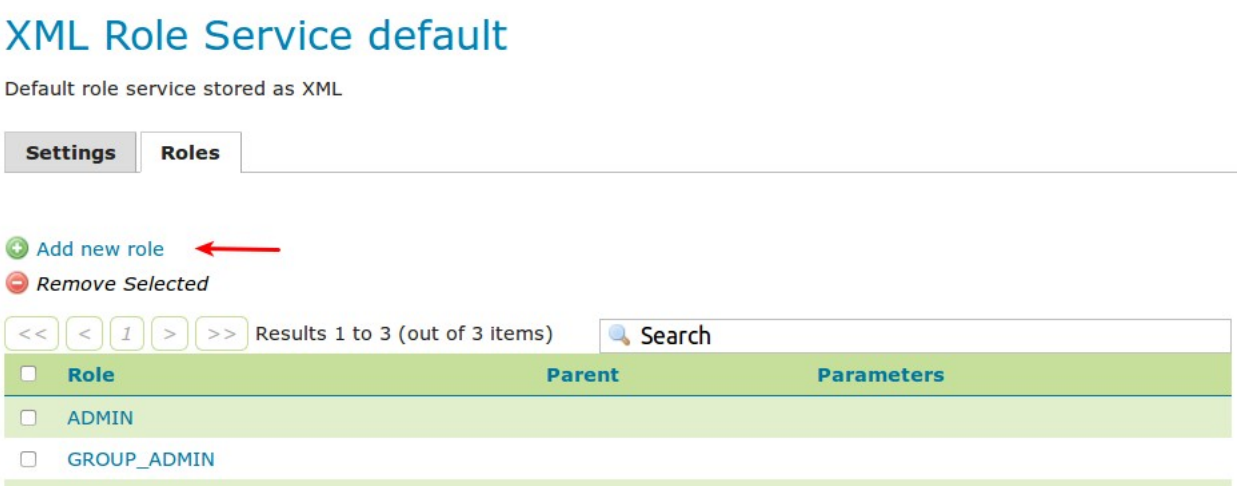
1. Click the `Users, Groups, and Roles` link located under the `Security` section of the navigation sidebar.

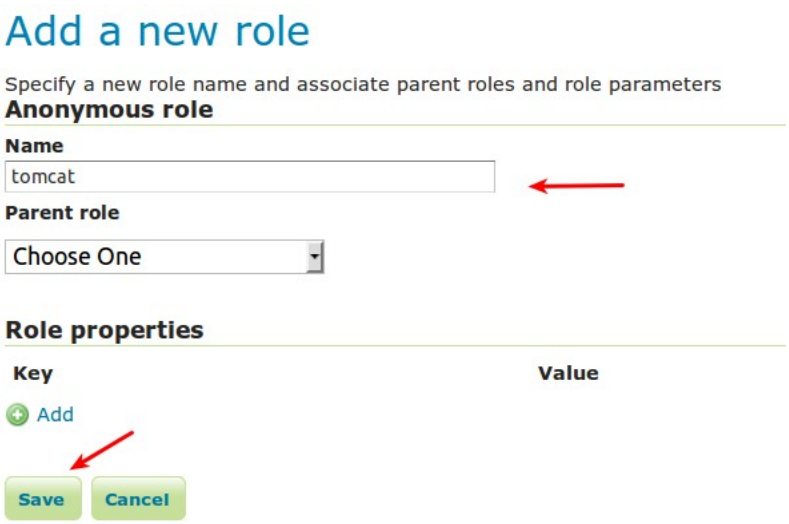
2. Click on `default` to work with the role service named "default".



3. Click on the `Roles` tab.
4. Click on the `Add new role` link.
   - Set the `Name` to "tomcat"
5. Save


XML Role Service default
Default role service stored as XML
Settings
Roles
Name
default


XML Role Service default
Default role service stored as XML
Settings
Roles
Add new role
Remove Selected
<< < 1 > >> Results 1 to 3 (out of 3 items)
Search
Role
Parent
Parameters
ADMIN
GROUP_ADMIN


Add a new role
Specify a new role name and associate parent roles and role parameters
Anonymous role
Name
tomcat
Parent role
Choose One
Role properties
Key
Value
Add
Save
Cancel

Configure Tomcat for authentication

By default Tomcat does not require authentication for web applications. In this section Tomcat will be configured to secure GeoServer requiring a basic authentication login.

1. Shut down Tomcat.
2. Edit the `conf/tomcat-users.xml` under the Tomcat root directory and add a user named "admin":

```
<user username="admin" password="password" roles="tomcat"/>
```

3. Edit the GeoServer `web.xml` file located at `webapps/geoserver/WEB-INF/web.xml` under the Tomcat root directory and add the following at the end of the file directly before the closing `</web-app>` element:

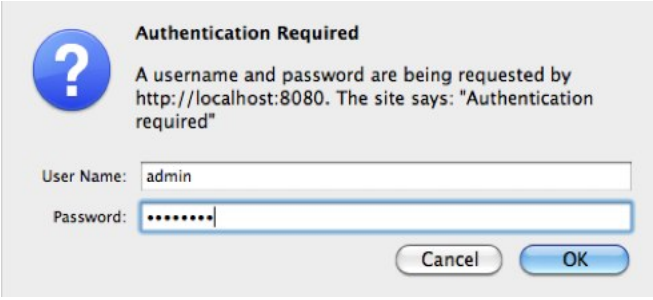
```
<security-constraint>
  <web-resource-collection>
    <url-pattern>/*</url-pattern>
     <http-method>GET</http-method>
     <http-method>POST</http-method>
  </web-resource-collection>
  <auth-constraint>
    <role-name>tomcat</role-name>
  </auth-constraint>
</security-constraint>

<login-config>
  <auth-method>BASIC</auth-method>
</login-config>
```

4. Save `web.xml` and restart Tomcat.

Note: It is necessary to add all the role names specified in the `web.xml` to the configured role service. This is duplicate work but there is currently no other solution.

Test J2EE login

1. Navigate to the GeoServer web admin interface. The result should be a prompt to authenticate.
2. Enter in the username "admin" and password "password"



The result should be the admin user logged into the GeoServer web admin.

13.10.6 Configuring HTTP Header Proxy Authentication

Introduction

Proxy authentication is used in multi-tier system. The user/principal authenticates at the proxy and the proxy provides the authentication information to other services.

This tutorial shows how to configure GeoServer to accept authentication information passed by HTTP header attribute(s). In this scenario GeoServer will do no actual authentication itself.

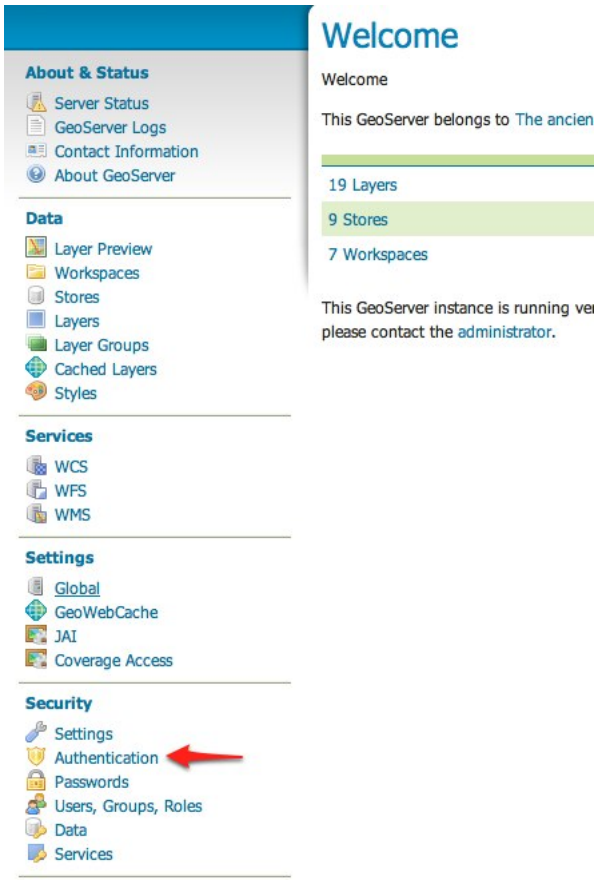
Prerequisites

This tutorial uses the curl utility to issue HTTP request that test authentication. Install curl before proceeding.

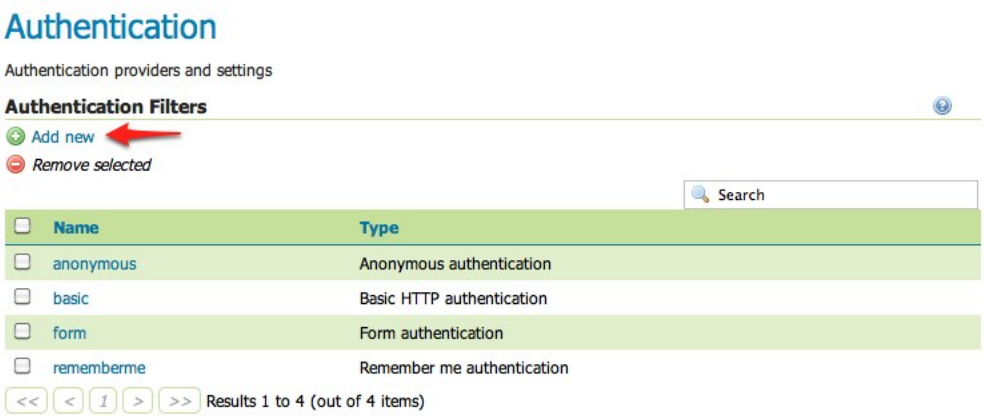
Note: Any utility that supports setting HTTP header attributes can be used in place of curl.

Configure the HTTP header filter

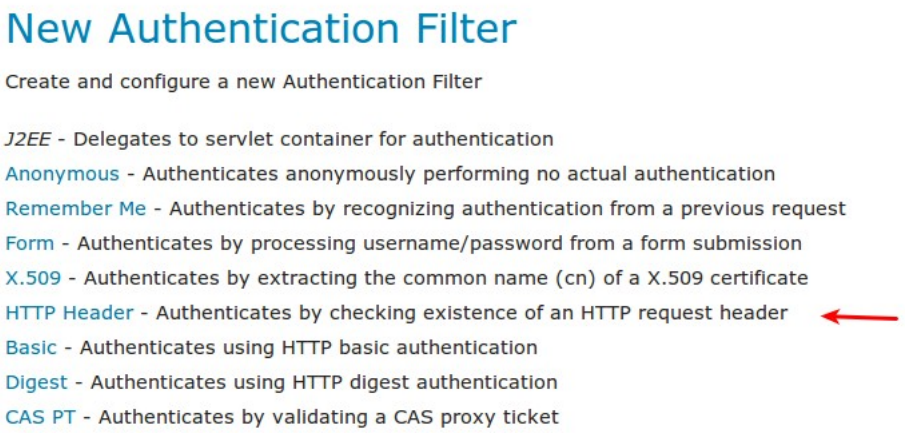
1. Start GeoServer and login to the web admin interface as the `admin` user.
2. Click the `Authentication` link located under the `Security` section of the navigation sidebar.



3. Scroll down to the `Authentication Filters` panel and click the `Add new` link.



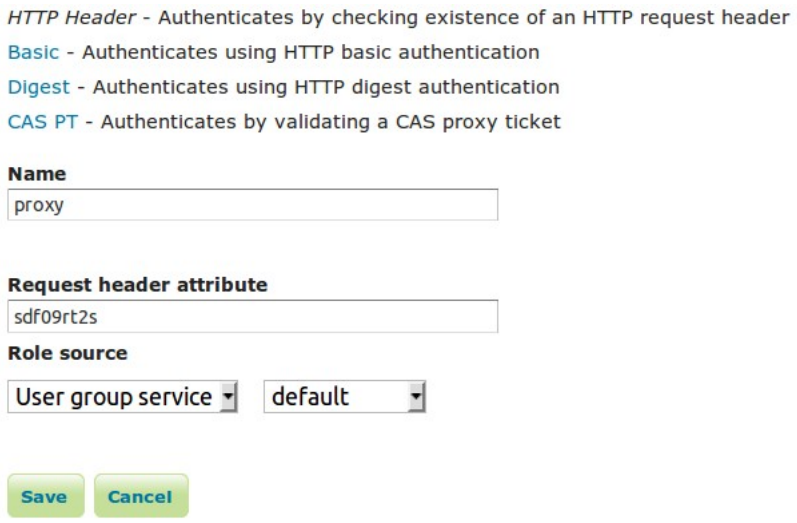
4. Click the `HTTP Header` link.



5. Fill in the fields of the settings form as follows:
   - Set `Name` to "proxy"
   - Set `Request header attribute to` to "sdf09rt2s"
   - Set `Role source` to "User group service"
   - Set the name of the user group service to "default"

Additional information about role services is here *Role source and role calculation*

> **Warning:** The tutorial uses the obscure "sdf09rt2s" name for the header attribute. Why not use "user" or "username" ?. In a proxy scenario a relationship of trust is needed between the proxy and GeoServer. An attacker could easily send an HTTP request with an HTTP header attribute "user" and value "admin" and operate as an administrator.
>
> One possibility is to configure the network infrastructure preventing such requests from all IP addresses except the IP of the proxy.



> This tutorial uses a obscure header attribute name which should be a shared secret between the proxy and GeoServer. Additionally, the use of SSL is recommended, otherwise the shared secret is transported in plain text.

1. Save.
2. Back on the authentication page scroll down to the `Filter Chains` panel.
3. Select "Default" from the `Request type` drop down.
4. Unselect the `basic` filter and select the `proxy` filter. Position the the `proxy` filter before the `anonymous` filter.

5. Save.

Secure OGC service requests

In order to test the authentication settings configured in the previous section a service or resource must be first secured. The `Default` filter chain is the chain applied to all OGC service requests so a service security

rule must be configured.

1. From the GeoServer home page and click the `Services` link located under the `Security` section of the navigation sidebar.

2. On the Service security page click the `Add new rule` link and add a catch all rule that secures all OGC service requests requiring the `ADMIN` role.

3. Save.

Test a proxy login

1. Execute the following curl command:

```
curl -v  -G "http://localhost:8080/geoserver/wfs?request=getcapabilities"
```

The result should be a 403 response signaling that access is denied. The output should look something like the following:

```
* About to connect() to localhost port 8080 (#0)
*   Trying ::1... connected
> GET /geoserver/wfs?request=getcapabilities HTTP/1.1
> User-Agent: curl/7.22.0 (x86_64-pc-linux-gnu) libcurl/7.22.0 OpenSSL/1.0.1 zlib/
↪1.2.3.4 libidn/1.23 librtmp/2.3
> Host: localhost:8080
> Accept: */*
>
< HTTP/1.1 403 Access Denied
< Content-Type: text/html; charset=iso-8859-1
< Content-Length: 1407
< Server: Jetty(6.1.8)
<
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1"/>
<title>Error 403 Access Denied</title>
</head>
...
```

2. Execute the same command but specify the `--header` option.:

```
curl -v --header "sdf09rt2s: admin" -G "http://localhost:8080/geoserver/wfs?
↪request=getcapabilities"
```

The result should be a successful authentication and contain the normal WFS capabilities response.

13.10.7 Configuring Apache HTTPD Session Integration

Introduction

When using Apache HTTPD as a proxy frontend for GeoServer, it is possible to share authentication with a proper configuration of both.

This requires enabling Session for the GeoServer location in Apache HTTPD and adding a custom Request Header with the session content, so that the GeoServer security system can receive user credentials and use them to authenticate the user with its internal filters.

To properly parse the received credentials we need to use the *Credentials From Request Headers* Authentication Filter.

Please note that the header containing the password is not sent back and forth to the user browser, but only from Apache HTTPD to GeoServer, so the password is not sent in clear through the public network.

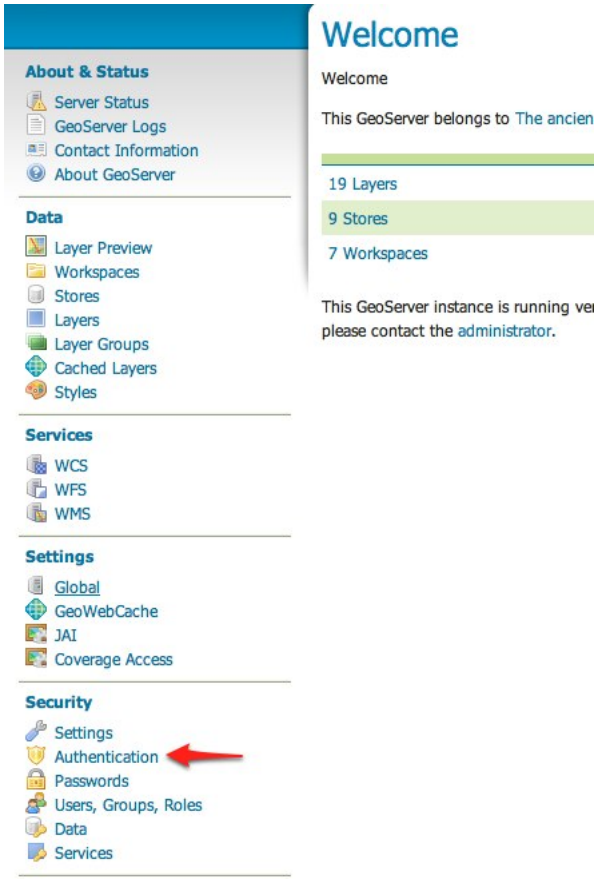
This tutorial shows how to configure GeoServer to read user credentials from the Apache HTTPD Session and use them for authentication purposes.

Prerequisites

This tutorial uses the curl utility to issue HTTP request that test authentication. Install curl before proceeding.

Configure the Credentials From Request Headers filter

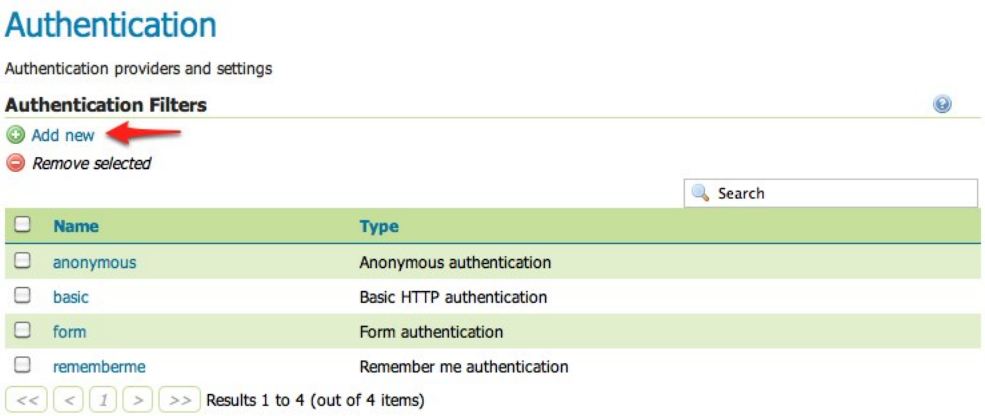
1. Start GeoServer and login to the web admin interface as the `admin` user.
2. Click the `Authentication` link located under the `Security` section of the navigation sidebar.

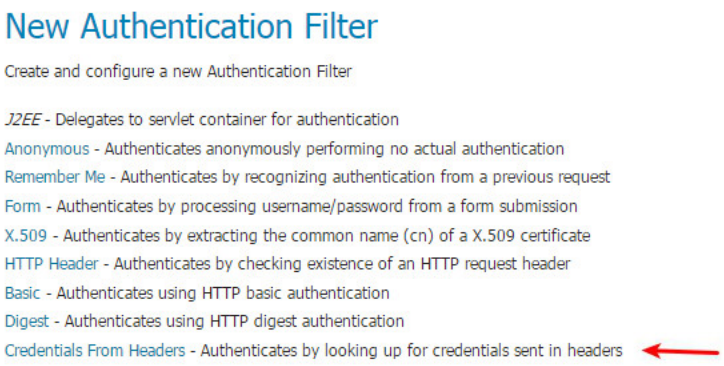


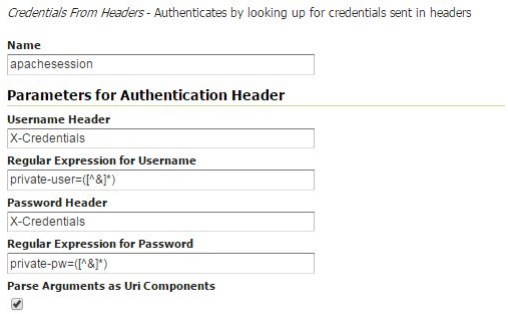
3. Scroll down to the `Authentication Filters` panel and click the `Add new` link.

4. Click the `Credentials From Headers` link.

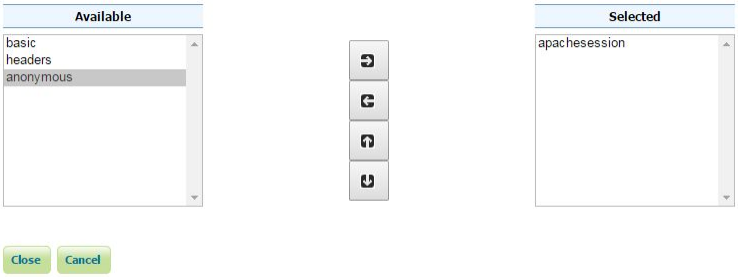
5. Fill in the fields of the settings form as follows:
   - Set `Name` to "apachessesion"
   - Set `Username Header` to "X-Credentials"
   - Set `Regular Expression for Username` to "private-user=([^&]*)"
   - Set `Password Header` to "X-Credentials"
   - Set `Regular Expression for Password` to "private-pw=([^&]*)"
6. Save.
7. Back on the authentication page scroll down to the `Filter Chains` panel.
8. Click on "default" in the chain grid.


Authentication
Authentication providers and settings
Authentication Filters
Add new
Remove selected
Search
Name
Type
anonymous
Anonymous authentication
basic
Basic HTTP authentication
form
Form authentication
rememberme
Remember me authentication
<< < 1 > >> Results 1 to 4 (out of 4 items)


New Authentication Filter
Create and configure a new Authentication Filter
J2EE - Delegates to servlet container for authentication
Anonymous - Authenticates anonymously performing no actual authentication
Remember Me - Authenticates by recognizing authentication from a previous request
Form - Authenticates by processing username/password from a form submission
X.509 - Authenticates by extracting the common name (cn) of a X.509 certificate
HTTP Header - Authenticates by checking existence of an HTTP request header
Basic - Authenticates using HTTP basic authentication
Digest - Authenticates using HTTP digest authentication
Credentials From Headers - Authenticates by looking up for credentials sent in headers


Credentials From Headers - Authenticates by looking up for credentials sent in headers
Name
apachesession
Parameters for Authentication Header
Username Header
X-Credentials
Regular Expression for Username
private-user=([^&]*)
Password Header
X-Credentials
Regular Expression for Password
private-pw=([^&]*)
Parse Arguments as Uri Components

9. Scroll down and remove all filters from the `Selected` list and add the `apachessesion` filter.



10. Close.
11. Save.

Test a login

1. Execute the following curl command (with a wrong password):

```
curl -v -H "X-Credentials: private-user=admin&private-pw=wrong" "http://
↪localhost:8080/geoserver/wms?service=WMS&version=1.1.1&request=GetCapabilities"
```

The result should be a 403 response signaling that access is denied. The output should look something like the following:

```
* About to connect() to localhost port 8080 (#0)
*   Trying ::1... connected
> GET /geoserver/wfs?request=getcapabilities HTTP/1.1
> User-Agent: curl/7.22.0 (x86_64-pc-linux-gnu) libcurl/7.22.0 OpenSSL/1.0.1 zlib/
↪1.2.3.4 libidn/1.23 librtmp/2.3
> Host: localhost:8080
> Accept: */*
>
< HTTP/1.1 403 Access Denied
< Content-Type: text/html; charset=iso-8859-1
< Content-Length: 1407
< Server: Jetty(6.1.8)
<
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1"/>
<title>Error 403 Access Denied</title>
</head>
...
```

2. Execute the same command but specify the right password.:

```
curl -v -H "X-Credentials: private-user=admin&private-pw=geoserver" "http://
↪localhost:8080/geoserver/wms?service=WMS&version=1.1.1&request=GetCapabilities"
```

The result should be a successful authentication and contain the normal WMS capabilities response.

Configure Apache HTTPD to forward an Header with authentication credentials

This can be done with an HTTPD configuration that looks like the following:

```
<Location  /geoserver>
        Session On
        SessionEnv On
        SessionHeader X-Replace-Session
        SessionCookieName session path=/
        SessionCryptoPassphrase secret
        RequestHeader set X-Credentials "%{HTTP_SESSION}e"
</Location>
```

This configuration adds a new *X-Credentials* Request Header to each GeoServer request. The request header will contain the HTTPD Session information in a special format.

An example of the Session content is the following:

> X-Credentials: private-user=admin&private-pw=geoserver

As you can see it contains both the username and password of the user, so the data can be used to authenticate the user in GeoServer.

13.10.8 Authentication with CAS

This tutorial introduces GeoServer CAS support and walks through the process of setting up authentication against a CAS server. It is recommended that the *Authentication chain* section be read before proceeding.

CAS server certificates

A running CAS server is needed.

The first step is to import the server certificates into the the GeoServer JVM.

If you need to export the *CRT* from the CAS server, you must execute the following command on the server JVM:

```
keytool -export -alias <server_name> -keystore <cas_jvm_keystore_path> -file server.
↪crt
```

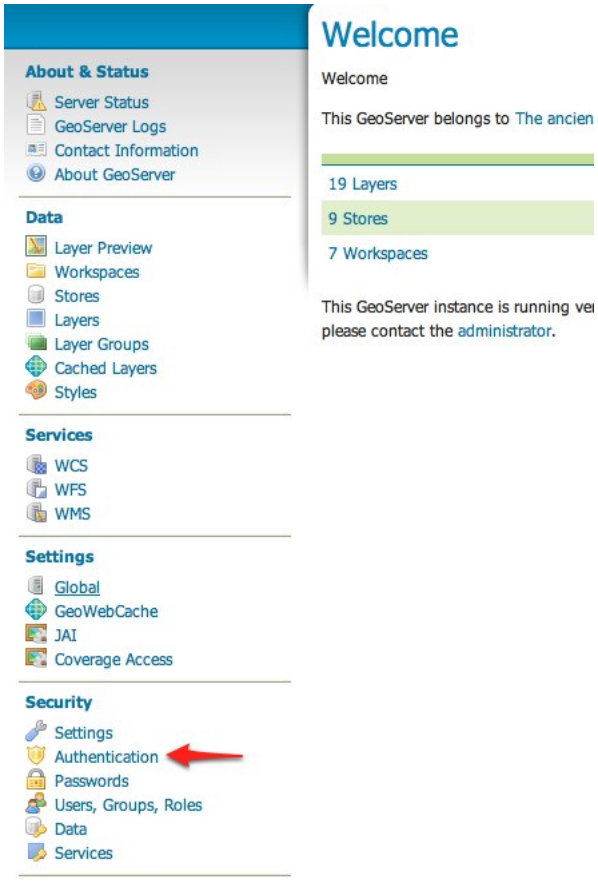
Once you have the *server.crt* file, the procedure to import the certificate into the JVM is the following one:

```
keytool -import -trustcacerts -alias <server_name> -file server.crt -keystore <path_
↪to_JRE_cacerts>
```

Enter the keystore password and confirm the certificate to be trustable.

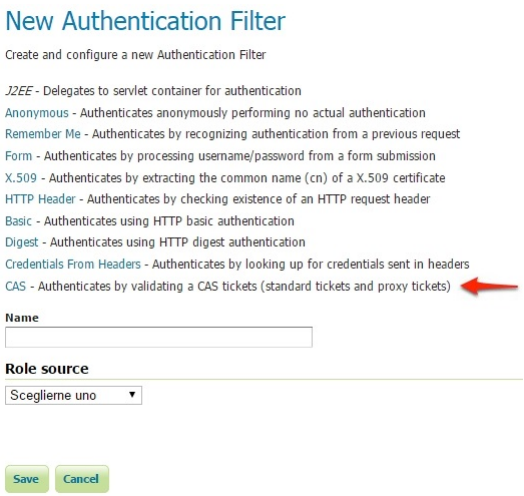
Configure the CAS authentication provider

1. Start GeoServer and login to the web admin interface as the `admin` user.
2. Click the `Authentication` link located under the `Security` section of the navigation sidebar.
3. Scroll down to the `Authentication Filters` panel and click the `Add new` link.

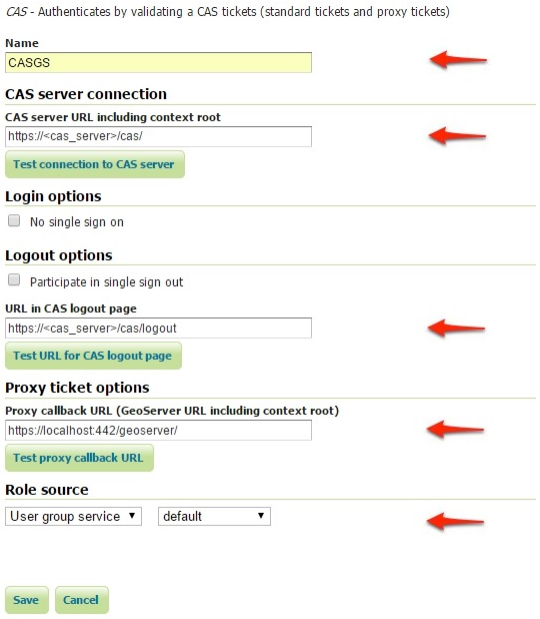

Welcome
About & Status
Server Status
GeoServer Logs
Contact Information
About GeoServer
Data
Layer Preview
Workspaces
Stores
Layers
Layer Groups
Cached Layers
Styles
Services
WCS
WFS
WMS
Settings
Global
GeoWebCache
JAI
Coverage Access
Security
Settings
Authentication
Passwords
Users, Groups, Roles
Data
Services
Welcome
This GeoServer belongs to The ancien
19 Layers
9 Stores
7 Workspaces
This GeoServer instance is running ver
please contact the administrator.


Authentication
Authentication providers and settings
Logout settings
Redirect URL after logout (empty,absolute or relative to context root)
/web/
SSL settings
SSL Port (default is 443)
443
Authentication Filters
Add new
Remove selected

4. Click the `CAS` link.



5. Fill in the fields of the settings form as follows:

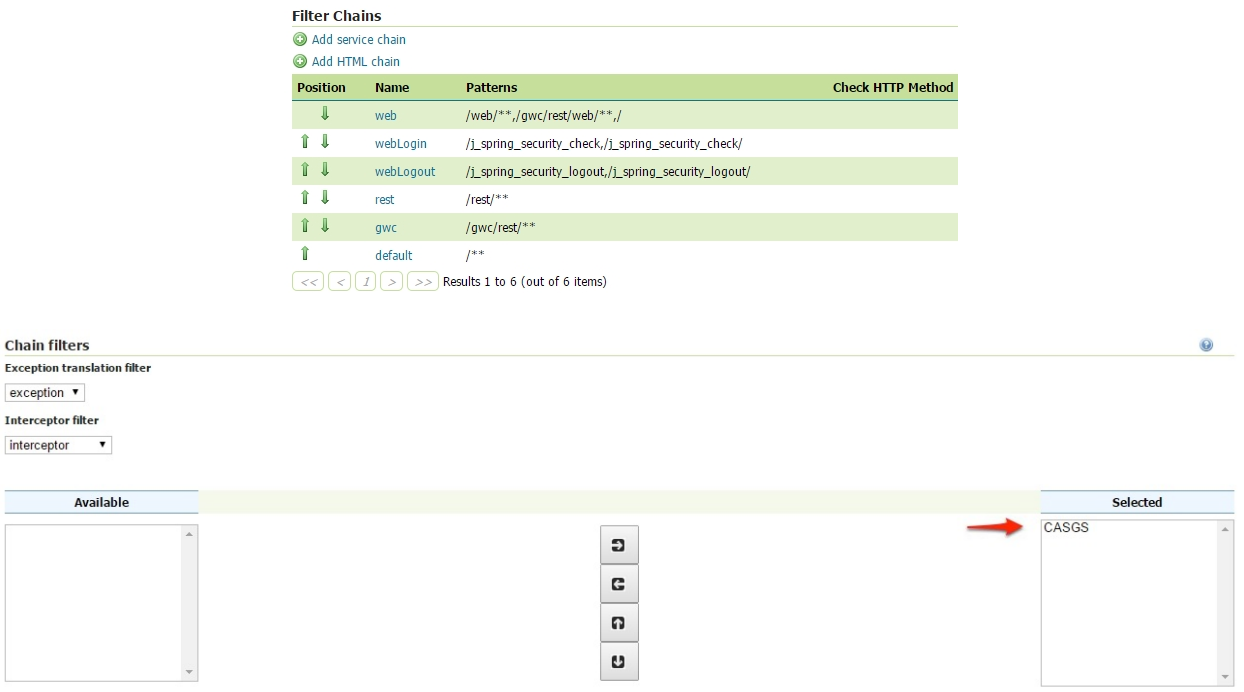


6. Update the filter chains by adding the new CAS filter.
7. Select the CAS Filter for each filter chain you want to protect with CAS.

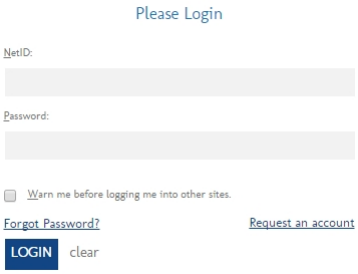
   Be sure to select and order correctly the CAS Filter.
8. Save.

Test a CAS login

1. Navigate to the GeoServer home page and log out of the admin account.



2. Try to login again, you should be able now to see the external CAS login form.



CHAPTER 14

GeoWebCache

GeoWebCache is a tiling server. It runs as a proxy between a map client and map server, caching (storing) tiles as they are requested, eliminating redundant request processing and thus saving large amounts of processing time. GeoWebCache is integrated with GeoServer, though it is also available as a standalone product for use with other map servers.

This section will discuss the version of GeoWebCache integrated with GeoServer. The first part will be show how GeoWebCache can be configured through the web admin interface, followed by a detailed discussion of the concepts of the.

For information about the standalone product, please see the GeoWebCache homepage.

14.1 GeoWebCache settings

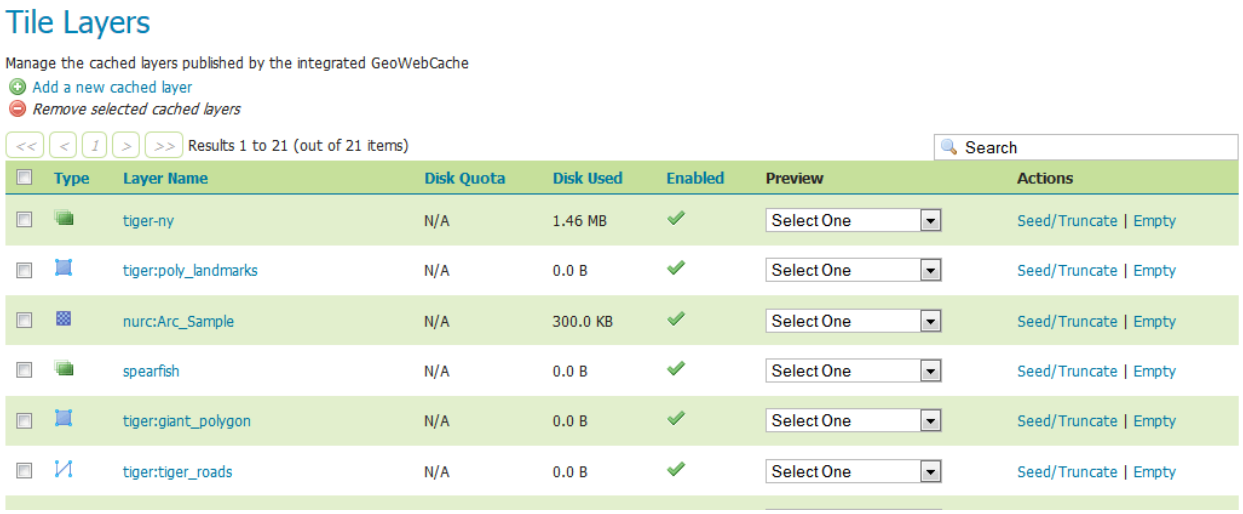
This section of the *Web administration interface* describes how to configure the tile caching options for GeoServer. GeoServer uses GeoWebCache to provide direct and integrated tile caching, and can dramatically increase your server's responsiveness and reliability.

The pages in this menu can be accessed on the left side of the screen under the heading *Tile Caching*.

Fig. 14.1: Tile Caching menu

14.1.1 Tile Layers

This page shows a listing of all of the layers known to the integrated GeoWebCache. It is similar to the *Layer Preview* for GeoWebCache, with many of the same options.



Note: There is also a link to the *GeoWebCache standalone demo page*.

Layer information

For each layer cached by GeoWebCache, the following information is available.

Disk Quota

The maximum amount of disk space that can be used for this layer. By default, this will be set to *N/A* (unbounded) unless *Disk Quotas* are enabled.

Disk Used

The current disk space being used by tiles for this particular layer.

Note: This counter will only be updated if disk quotas are enabled. If disk quotas are not enabled, tiles will still be saved to disk, but the counter will remain as `0.0 B`.

Enabled

Indicates whether tile caching is enabled for this layer. It is possible to have a layer definition here but to not have tile caching enabled (set in the layer properties).

Preview

Similar to *Layer Preview*, this will generate a simple OpenLayers application populated with tiles from one of the available gridset/image format combinations. Select the desired option from the menu to view in OpenLayers.

Seed/Truncate

Opens the GeoWebCache page for automatically seeding and truncating the tile cache. Use this if you want to pre-populate some of your cache.

Empty

Will remove all saved tiles from the cache. This is identical to a full truncate operation for the layer.

Add or remove cached layers

The list of layers displayed on this page is typically the same as, or similar to, the full list of layers known to GeoServer. However, it may not be desirable to have every layer published in GeoServer have a cached layer component. In this case, simply select the box next to the layer to remove, and click *Remove selected cached layers*. The layer will be removed from GeoWebCache, and the disk cache for this layer will be entirely removed.

> **Warning:** Deleting the tile cache cannot be undone.

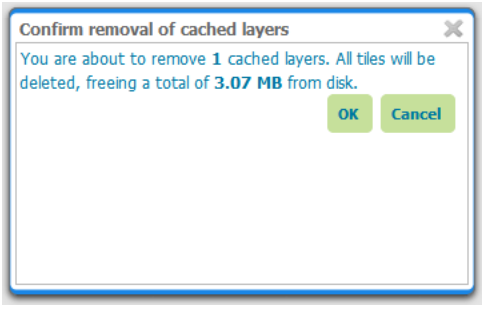


Fig. 14.2: Removing a cached layer

To add in a layer from GeoServer (if it wasn't set up to be added automatically), click the *Add a new cached layer* link.

Configuring a cached layer

You have two options for layer configuration. The first option is to load the layer using the default (global) settings. To do this, select the layer you wish to start caching, and click the *Configure selected layers with caching defaults* link. The second option is to configure the caching parameters manually, via the *layer configuration* pages. To do this, just click the layer name itself.

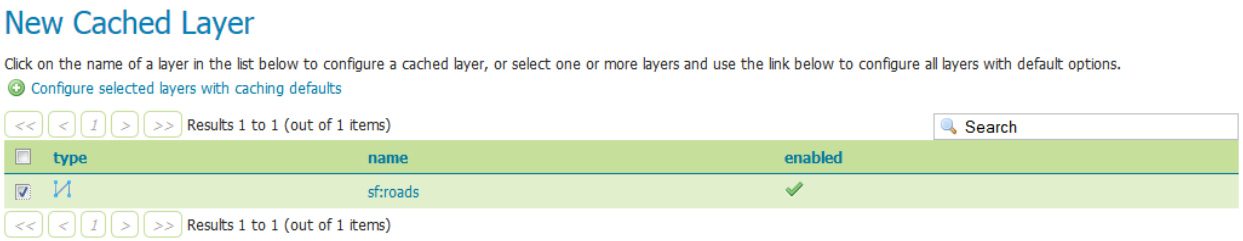


Fig. 14.3: Adding a new cached layer

Parameter Filters

Parameter filters allow GeoWebCache to cache a layer with varying parameters such as `STYLES`, `TIME`. One is needed for each parameter to be cached and it needs to know how to recognize valid values to be cached and which values are the same as other values so they only get cached once. There are several different kinds of filter as a result.

Adding a Filter

At the bottom of the filter list in the text box beside *Add filter* specify the name of the parameter. In the drop down box select the kind of filter you want then click the button. For a filter that automatically tracks the layers styles in a parameter named `STYLES` click the *Add Style Filter* button.

Removing a Filter

To remove a filter, click the button to the right of the filter's entry in the filter list.

Types of filter

All parameter filters take a default parameter that will be used if the parameter was not specified. Specific types of parameter filter provide different ways of specifying which parameter values are allowed, and which are equivalent to one another and should be cached together.

List of Strings

The `stringParameterFilter` takes a collection of plain text strings. If the value matches one of the strings, it is valid, otherwise it is not. Matching can be done in a case sensitive way, or the strings can all be converted to upper or lower case before matching. As case rules vary between languages, the locale to use for case changes can be specified.

Regular Expression

The *regexParameterFilter* takes a regular expression to match strings. This should be used with caution as it potentially allows an arbitrarily large number of caches to be created. Like the string filter, it can be normalized for case.

List of Numbers

The `floatParameterFilter` is like the string filter in taking a list of values, but it uses floating point numbers rather than arbitrary text strings. A threshold can be given to pull close numbers to a standard value.

List of Whole Numbers

The `integerParameterFilter` is like the float filter but works with integer/whole number values.

Styles

The `styleParameterFilter` is connected to the GeoServer catalog and knows what styles are available for the layer and when they change. You can specify a default distinct from the normal layer default if you wish, and restrict the range of additional styles available if you do not wish to cache all of them.

14.1.2 Demo page

In addition to the *Tile Layers* page, there is also a demo page where you can view configured layers, reload the configuration (when changing settings or adding new layers), and seed or refresh the existing cache on a per-layer basis.

As this interface is part of the standalone GeoWebCache, some of the functionality here is duplicated from the *Tile Layers* page.

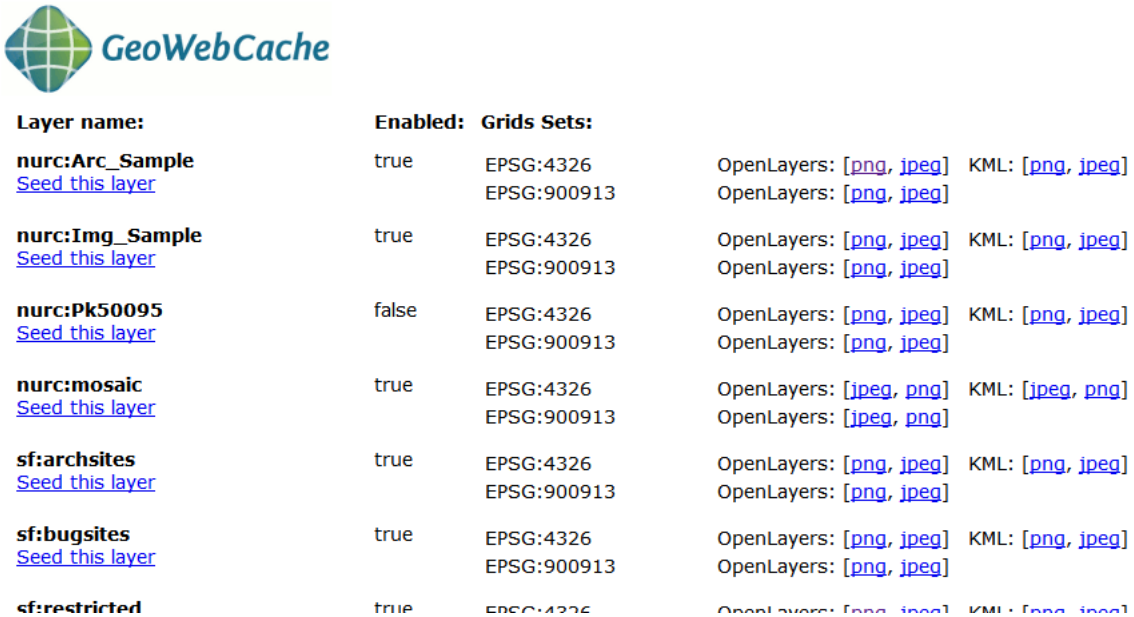


Fig. 14.4: Built-in demo page

Viewing

To view the demo page, append `/gwc/demo` to the address of your GeoServer instance. For example, if your GeoServer is at the following address:

```
http://localhost:8080/geoserver
```

The GeoWebCache demo page is accessible here:

```
http://localhost:8080/geoserver/gwc/demo
```

If there is a problem loading this page, verify the steps on the *Using GeoWebCache* page have been carried out successfully.

Reload configuration

The demo page contains a list of every layer that GeoWebCache is aware of. This is typically (though not necessarily) identical to the list of layers as published in the GeoServer WMS capabilities document. If configuration changes are made to GeoServer, GeoWebCache will not automatically become aware of them. To ensure that GeoWebCache is using the latest configuration information, click the **Reload Configuration** button. Reloading the configuration will trigger authentication to GeoServer, and will require an administration username and password. Use the same username and password that you would use to log on to the *Web administration interface*. After a successful logon, the number of layers found and loaded will be displayed.

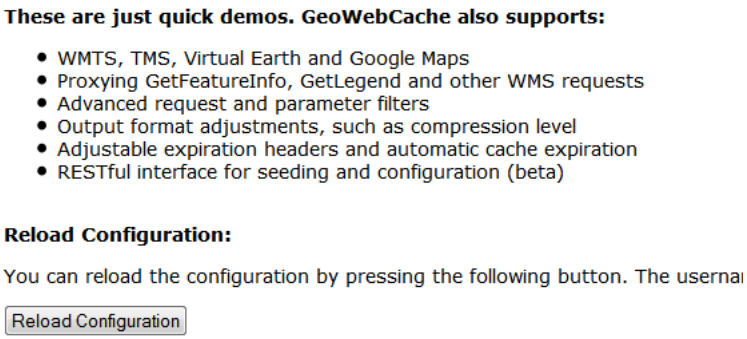


Fig. 14.5: Reloading the configuration

Layers and output formats

For each layer that GeoWebCache serves, links are typically available for a number of different projections and output formats. By default, **OpenLayers** applications are available using image formats of PNG, PNG8, GIF, and JPEG in both **EPSG:4326** (standard lat/lon) and **EPSG:900913** (used in Google Maps) projections. In addition, **KML output** is available (EPSG:4326 only) using the same image formats, plus vector data ("kml").

Also on the list is an option to seed the layers (*Seed this layer*). More on this option below.

Seeding

You can configure seeding processes via the *Web administration interface*. See the *Tile Layers* page for more information.

It is also possible to configure seeding process via the *Demo page*. The page contains a link next to each layer entitled *Seed this layer*. This link will trigger authentication with the GeoServer configuration. Use the same username and password that you would use to log on to the *Web administration interface*. After a successful logon, a new page shows up with seeding options.

The seeding options page contains various parameters for configuring the way that the layer is seeded.

Option	Description
`Number of threads to use`	Possible values are between **1** and **16**.
`Type of operation`	Sets the operation. There are three possible values: **Seed** (creates tiles, but does not overwrite existing ones), **Reseed** (like Seed, but overwrites existing tiles) and **Truncate** (deletes all tiles within the given parameters)
`SRS`	Specifies the projection to use when creating tiles (default values are **EPSG:4326** and **EPSG:900913**)
`Format`	Sets the image format of the tiles. Can be **application/vnd.google-earth.kml+xml** (Google Earth KML), **image/gif** (GIF), **image/jpeg** (JPEG), **image/png** (24 bit PNG), and **image/png8** (8 bit PNG)
`Zoom start`	Sets the minimum zoom level. Lower values indicate map views that are more zoomed out. When seeding, GeoWebCache will only create tiles for those zoom levels inclusive of this value and `Zoom stop`.
`Zoom stop`	Sets the maximum zoom level. Higher values indicate map views that are more zoomed in. When seeding, GeoWebCache will only create tiles for those zoom levels inclusive of this value and `Zoom start`.
`Bounding box`	*(optional)* Allows seeding to occur over a specified extent, instead of the full extent of the layer. This is useful if your layer contains data over a large area, but the application will only request tiles from a subset of that area. The four boxes correspond to **Xmin**, **Ymin**, **Xmax**, and **Ymax**.

Warning: Currently there is no progress bar to inform you of the time required to perform the operation, nor is there any intelligent handling of disk space. In short, the process may take a *very* long time, and the cache may fill up your disk. You may wish to set a *Disk quota* before running a seed job.

14.1.3 Caching defaults

The Caching Defaults page shows the global configuration options for the tile caching functionality in GeoServer, an embedded GeoWebCache.

GWC Provided Services

In addition to the GeoServer endpoints, GeoWebCache provides other endpoints for OGC services. For example, the GeoServer WMS endpoint is available at:

```
http://GEOSERVER_URL/wms?...
```

The GeoWebCache WMS endpoint is:

```
http://GEOSERVER_URL/gwc/service/wms?...
```

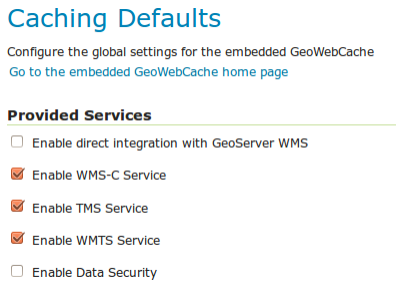


Fig. 14.6: Provided services

The following settings describe the different services that can be enabled with GeoWebCache.

Enable direct integration with GeoServer WMS

Direct integration allows WMS requests served through GeoServer to be cached as if they were received and processed by GeoWebCache. This provides all the advantages of using a tile server while still employing the more-flexible GeoServer WMS as a fallback. See the section on *Using GeoWebCache* for more details about this feature.

With direct integration, tile caching is enabled for all standard WMS requests that contain the `tiled=true` parameter and conform to all required parameters.

This setting is disabled by default. When enabling this option, it is a good idea to also turn on *Disk Quotas* as well, to prevent unbounded growth of the stored tiles.

Enable WMS-C Service

Enables the Cached Web Map Service (WMS-C) service. When this setting is enabled, GeoWebCache will respond to its own WMS-C endpoint:

```
http://GEOSERVER_URL/gwc/service/wms?SERVICE=WMS&VERSION=1.1.1&TILED=true&...
```

When the service is disabled, calls to the capabilities document will return a `Service is disabled` message.

Enable TMS Service

Enables the Tiled Map Service (TMS) endpoint in GeoWebCache. With the TMS service, GeoWebCache will respond to its own TMS endpoint:

```
http://GEOSERVER/URL/gwc/service/tms/1.0.0
```

When the service is disabled, calls to the capabilities document will return a `Service is disabled` message.

Enable WMTS Service

Enables the Web Map Tiled Service (WMTS) endpoint in GeoWebCache. When this setting is enabled, GeoWebCache will respond to its own WMTS endpoint:

```
http://GEOSERVER/URL/gwc/service/wmts?...
```

When the service is disabled, calls to the capabilities document will return a `Service is disabled` message.

HTTP RESTful API is available through the existing GWC integration allowing clients to retrieve the following resources:

- capabilities document
- tile
- feature info

For more information read GWC WMTS documentation.

Enable Data Security

Enables the *GeoServer Data Security* in the embedded GeoWebCache.

Default Caching Options for GeoServer Layers

This section describes the configuration of the various defaults and other global options for the tile cache in GeoServer.

Fig. 14.7: Default caching options

Automatically configure a GeoWebCache layer for each new layer or layer group

This setting, enabled by default, determines how layers in GeoServer are handled via the embedded GeoWebCache. When this setting is enabled, an entry in the GeoWebCache layer listing will be created whenever a new layer or layer group is published in GeoServer. Use this setting to keep the GeoWebCache catalog in sync. (This is enabled by default.)

Automatically cache non-default styles

By default, only requests using the default style for a given layer will be cached. When this setting is enabled, all requests for a given layer, even those that use a non-standard style will be cached. Disabling

this may be useful in situations where disk space is an issue, or when only one default style is important.

Default metatile size

A metatile is several tiles combined into a larger one. This larger metatile is generated and then subdivided before being served back (and cached) as standard tiles. The advantage of using metatiling is in situations where a label or geometry lies on a boundary of a tile, which may be truncated or altered. With metatiling, these tile edge issues are greatly reduced.

Moreover, with metatiling, the overall time it takes to seed the cache is reduced in most cases, when compared with rendering a full map with single tiles. In fact, using larger metatiling factors is a good way to reduce the time spent in seeding the cache.

The disadvantage of metatiling is that at large sizes, memory consumption can be an issue.

The size of the default metatile can be adjusted here. By default, GeoServer sets a metatile size of **4x4**, which strikes a balance between performance, memory usage, and rendering accuracy.

Default gutter size

The gutter size sets the amount of extra space (in pixels) used when generating a tile. Use this in conjunction with metatiles to reduce problems with labels and features not being rendered incorrectly due to being on a tile boundary.

Default Cache Formats

This setting determines the default image formats that can be cached when tiled requests are made. There are four image formats that can be used when saving tiles:

- PNG (24-bit PNG)
- PNG8 (8-bit PNG)
- JPEG
- GIF

The default settings are subdivided into vector layers, raster layers, and layer groups. You may select any of the above four formats for each of the three types of layers. Any requests that fall outside of these layer/format combinations will not be cached if sent through GeoServer, and will return an error if sent to the GeoWebCache endpoints.

These defaults can be overwritten on a per-layer basis when *editing the layer properties*.

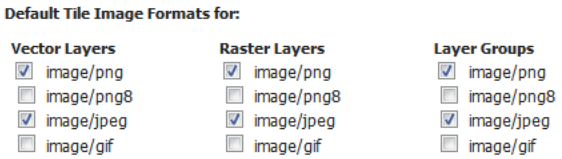


Fig. 14.8: Default image formats

In Memory BlobStore Options

These options are used for enabling/disabling In Memory Caching for GeoWebCache. This feature can be used for saving GWC tiles directly in memory, for a fast data retrieval.

Enable

This parameter allows to enable or disable in memory caching. By default it is disabled.

Avoid Persistence

This parameter can be used in order to avoid to save any file in the file system, keeping all the GWC tiles only in memory. By default it is disabled.

Available Caches

This parameter defines which Cache method can be used for In Memory Caching. By default the Guava Caching is used. Note that if a caching method requires an immutable configuration at GeoServer startup like HazelCast, the *Hard Memory limit*, *Eviction Policy*, *Eviction Time* and *Concurrency Level* parameters are disabled.

More informations on how to configure a new Cache object can be found in the GeoWebCache *Configuration* page.

Cache Hard Memory limit (Mb)

Parameter for configuring in memory cache size in MB.

Cache Eviction Policy

Parameter for configuring in memory cache eviction policy, it may be: LRU, LFU, EXPIRE_AFTER_WRITE, EXPIRE_AFTER_ACCESS, NULL

This eviction policies may not be supported by all caches implementations. For example, Guava Caching only supports the eviction policies: EXPIRE_AFTER_WRITE, EXPIRE_AFTER_ACCESS and NULL.

Note, only the eviction policies accepted by the selected cache will be shown on the UI.

Cache Eviction Time (in Seconds)

Paramter for configuring in memory cache eviction time. It is in Seconds.

Note: Note that this parameter is also used for configuring an internal thread which performs a periodical cache cleanup.

Cache Concurrency Level

Paramter for configuring in memory cache concurrency.

Clear In Memory Cache

Button for clearing all the tiles in the in-memory cache.

Cache Statistics

Various statistics parameters associated to the in memory cache.

Update Cache Statistics

Button for updating cache statistics seen above. The statistics are always related to the local cached entries, even in case of distributed in-memory caching

---

Note: Note that some Caches do not provide all the statistics parameters, in that case the user will only see *"Unavailable"* for those parameters.

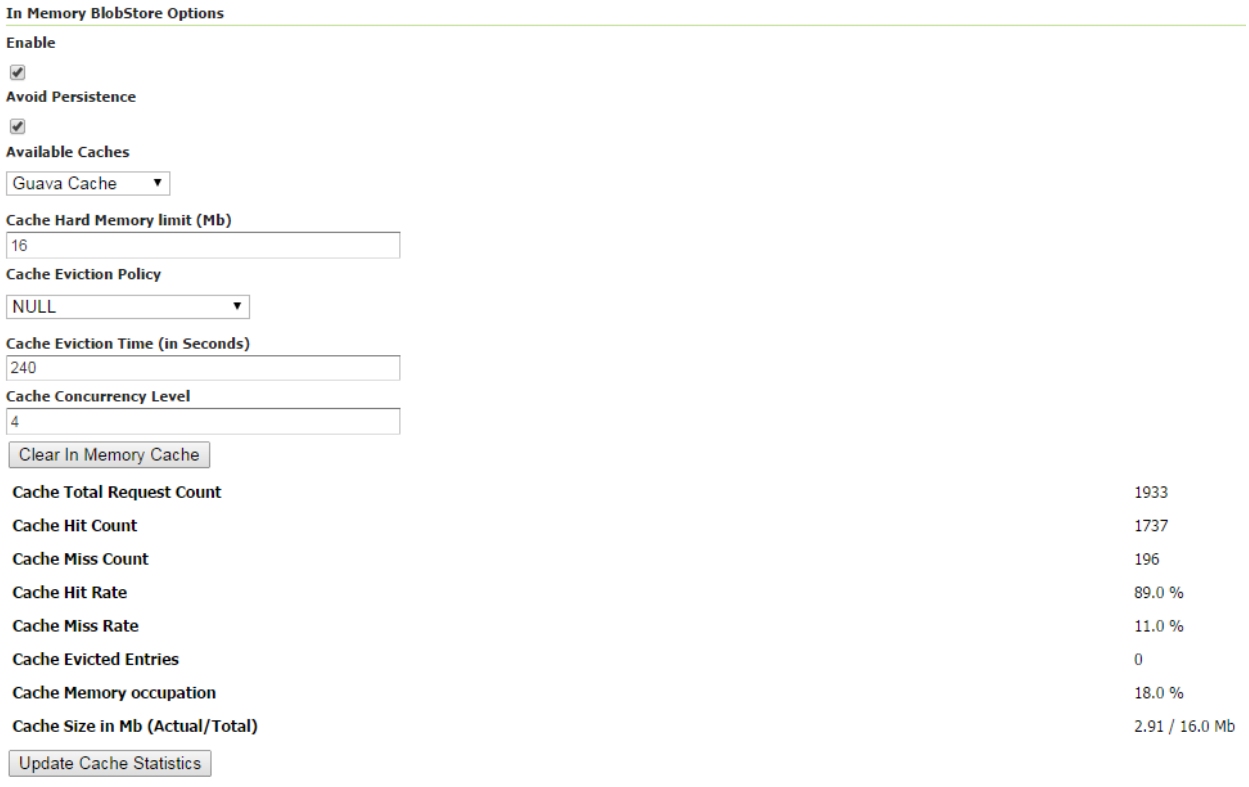


Fig. 14.9: *In Memory BlobStore Options*

Note: Note that in the *TileCaching* tab for each Layer, you may decide to disable in memory caching for the selected Layer by clicking on the **Enable In Memory Caching for this Layer** checkbox. This option is disabled for those cache which don't support this feature.

Default Cached Gridsets

This section shows the gridsets that will be automatically configured for cached layers. While there are some pre-configured gridsets available, only two are enabled by default. These correspond to the most common and universal cases:

- EPSG:4326 (geographic) with 22 maximum zoom levels and 256x256 pixel tiles
- EPSG:900913 (spherical Mercator) with 31 maximum zoom levels and 256x256 pixel tiles

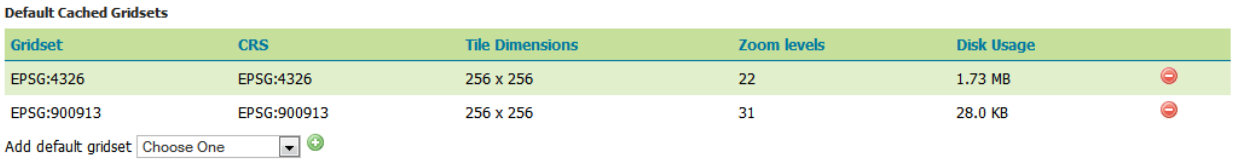


Fig. 14.10: *Default gridsets*

To add a pre-existing grid set, select it from the *Add default grid set* menu, and click the Add icon (green circle with plus sign).

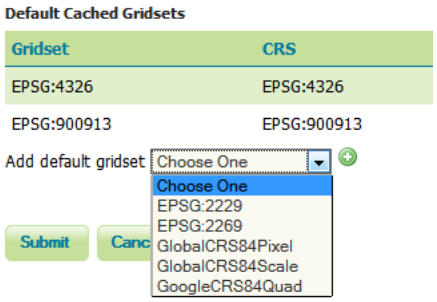


Fig. 14.11: *Adding an existing gridset to the list of defaults*

These definitions are described in more detail on the *Gridsets* page.

14.1.4 Gridsets

A gridset defines a spatial reference system, bounding box (extent), a list of zoom levels (resolutions or scale denominators), and tile dimensions. Tile requests must conform to the gridset matrix, otherwise caching will not occur.

This page allows you to edit existing saved gridsets or create new ones. There are five preconfuigred gridsets, all in one of two coordinate reference systems: EPSG:4326 and EPSG:900913. For additional CRS support, new gridsets can be created. Another reason to create a new gridset would be to set a different tile size or different number of zoom levels.

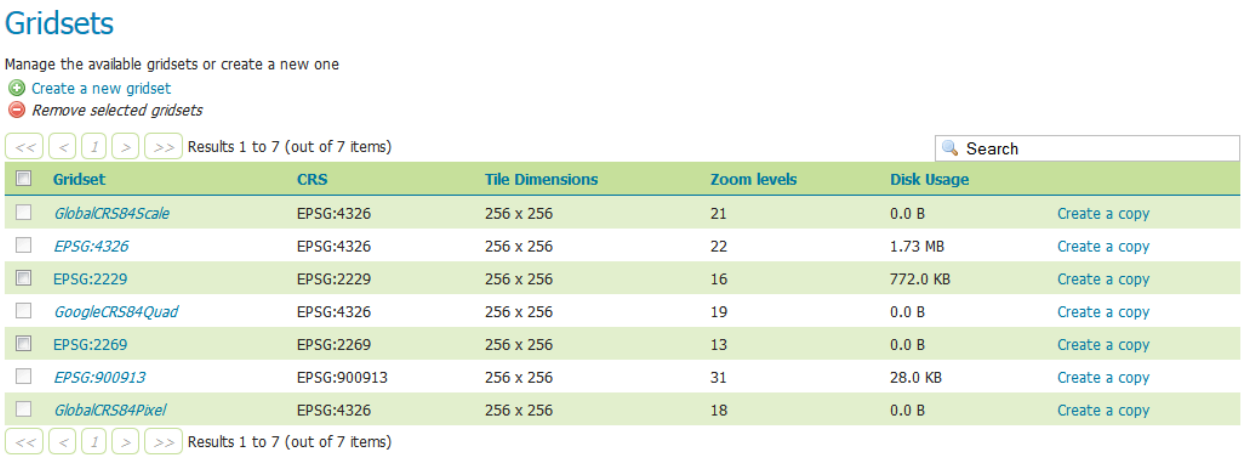


Fig. 14.12: Gridsets menu

Creating a new gridset

To create a new gridset, click *Create new gridset*. You will then be asked to enter a range of parameters.

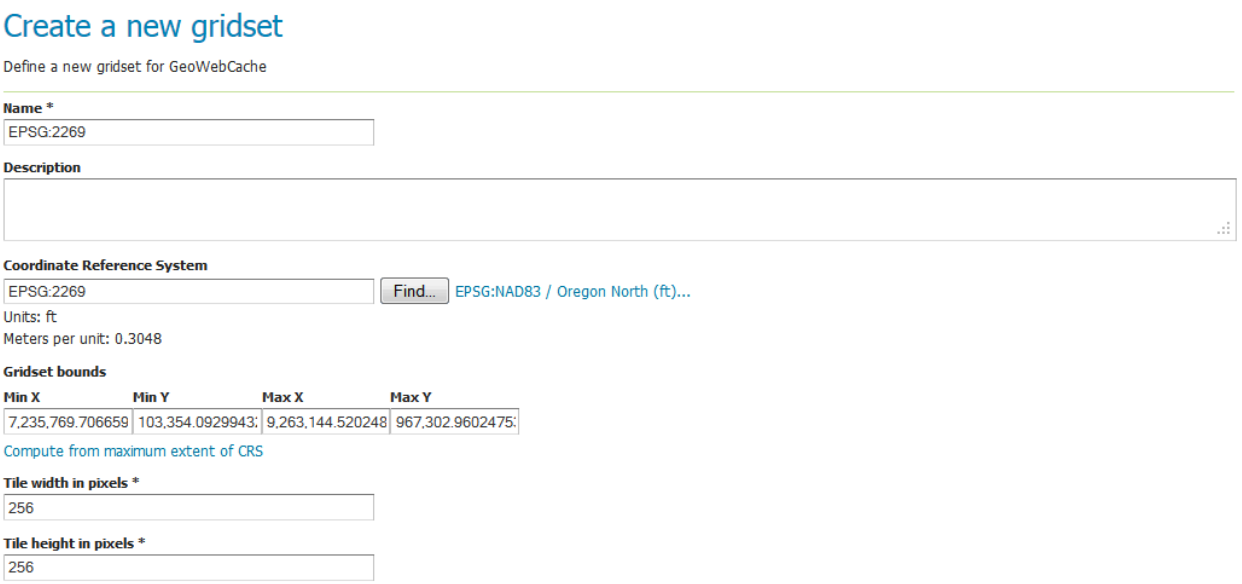


Fig. 14.13: Creating a new gridset

Name

The short name of the new gridset.

Description

Metadata on the gridset.

Coordinate Reference System

The Coordinate Reference System (CRS) to use in the gridset. You can select from any CRS that GeoServer recognizes. After selection, both the units (meters, feet, degrees, etc.) and the number of meters per unit will be displayed.

Gridset bounds

Sets the maximum extent for the gridset. Typically this is set to be the maximum extent of the CRS used, but a smaller value can be substituted if desired. To populate the max extent in the fields, click *Compute from maximum extent of CRS*.

Tile width and height

Sets the tile dimensions. Default is **256x256 pixels**. The tile dimensions can be anything from 16 to 2048 pixels. In addition, the tiles need not be square.

Tile matrix set

The tile matrix set (or tile pyramid) is a list of zoom levels containing ever increasing amounts of tiles. This three dimensional collection of tile "slots" creates the framework where actual image tiles will be saved. You can define the tile matrix based on resolutions or scale denominators.

Click *Add zoom level* to generate the first zoom level. The parameters will be automatically configured such that the full extent of will be contained by a single pixel's height. The number of pixels in a given zoom level will be displayed, along with the Pixel Size, Scale, and an optional Name, where you can give a name to each zoom level if desired.

Typically each additional zoom level is twice as large in each dimension, and so contains four times as many tiles as the previous zoom level. The actual values will be populated automatically during subsequent clicking of the *Add zoom level* link. These defaults are usually sufficient, and you need only determine the maximum number of zoom levels desired for this gridset.

When finished, click *Save*. Before you will be able to use this new gridset with a layer, you will need to add this gridset to the layer's list of available gridsets. This is done on an individual layer's *properties* page. You can also add this gridset to the default list on the *Caching defaults* page.

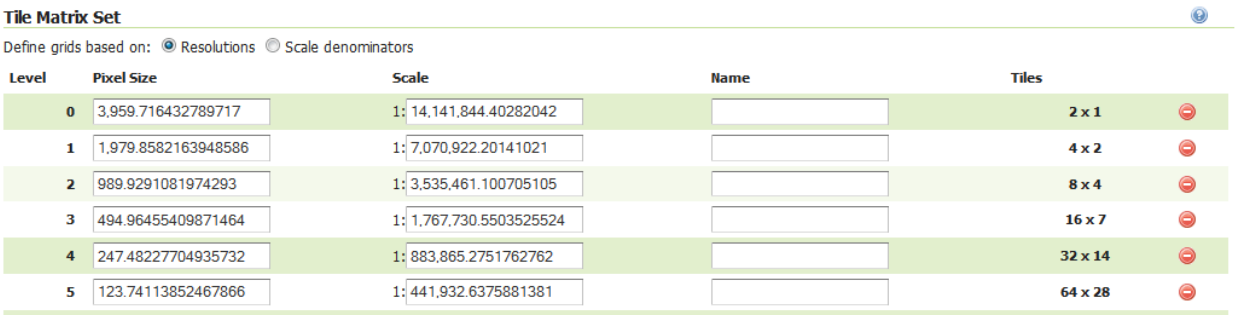


Fig. 14.14: Tile matrix set

Editing a gridset

Click an existing gridset to open it for editing. Please note that the built-in gridsets cannot be edited. They can, however, be copied.

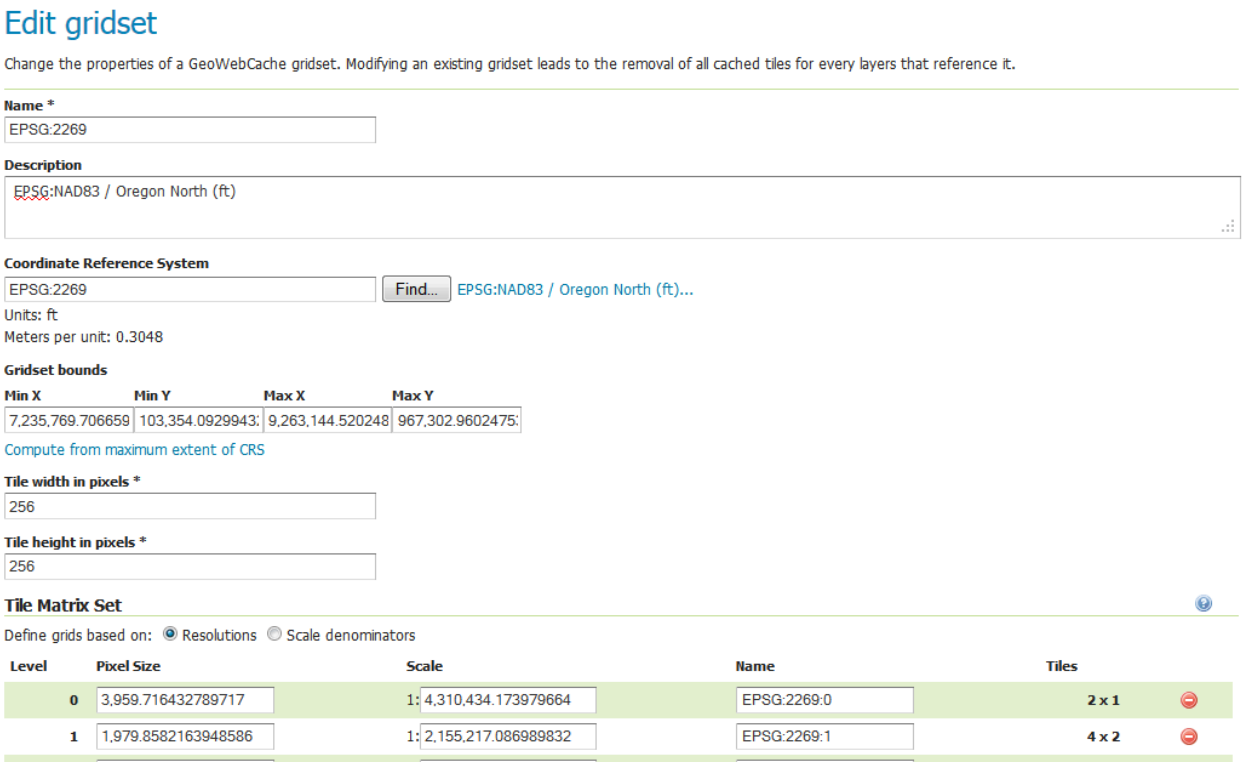


Fig. 14.15: Editing a gridset

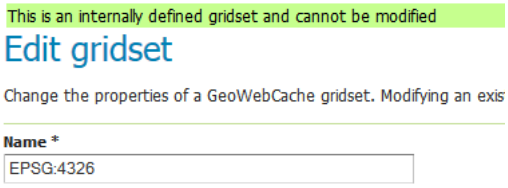


Fig. 14.16: This gridset is read-only

Copying a gridset

As there are many configuration options for a gridset, it is often more convenient to copy an existing gridset. For any of the existing gridsets, click the *Create a copy* link to copy the gridset information to a new gridset.

Removing a gridset

To remove a gridset, select the check box next to the gridset or gridsets, and click *Remove selected gridsets*.

Warning: Removing a gridset definition will remove not only the gridset definition, but also any tiles on any layers generated with this gridset.

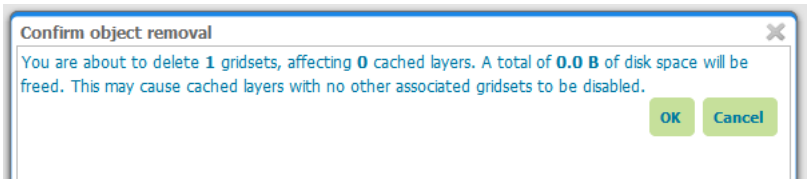


Fig. 14.17: Removing a gridset

14.1.5 Disk Quotas

The Disk Quotas page manages the disk usage for cached tiles and allows you to set the global disk quota. Individual layer quotas can be set in the layer's *properties* page.

By default, disk usage for cached tiles is unbounded. However, this can cause disk capacity issues, especially when using Direct WMS integration (see *Disk Quotas* for more on this). Setting a disk quota establishes disk usage limits.

When finished making any changes, remember to click *Submit*.

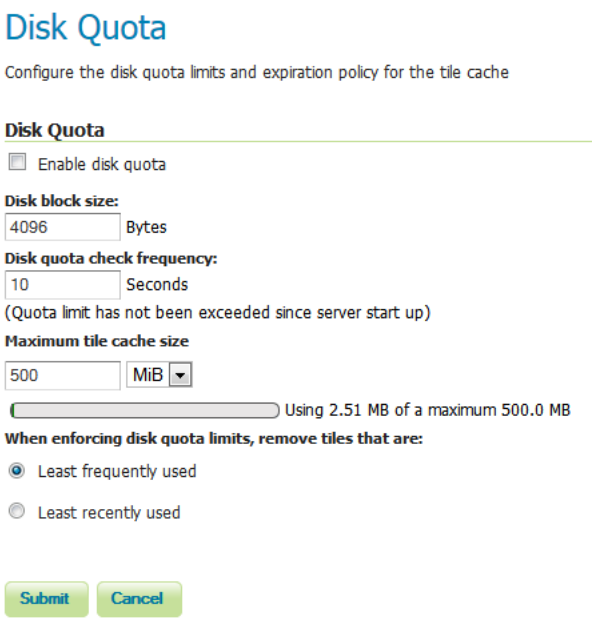


Fig. 14.18: Disk quota

Enable disk quota

When enabled, the disk quota will be set according to the options listed below. The setting is disabled by default.

Disk quota check frequency

This setting determines how often the cache is polled for any overage. Smaller values (more frequent polling) will slightly increase disk activity, but larger values (less frequent polling) may cause the disk quota to be temporarily exceeded. The default is **10 seconds**.

Maximum tile cache size

The maximum size for the cache. When this value is exceeded and the cache is polled, tiles will be removed according to the policy. Note that the unit options are **mebibytes (MiB)** (approx. 1.05MB), **gibibytes (GiB)** (approx. 1.07GB), and **tebibytes (TiB)** (approx. 1.10TB). Default is **500 MiB**.

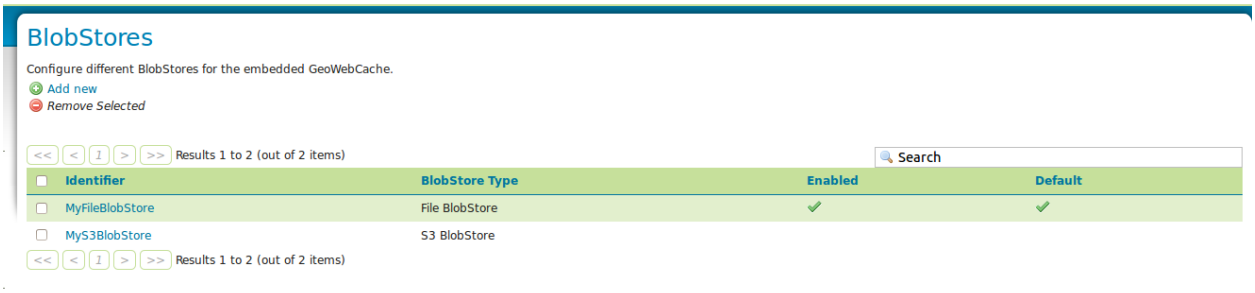
The graphic below this setting illustrates the size of the cache relative to the disk quota.

Tile removal policy

When the disk quota is exceeded, this policy determines how the tiles to be deleted are identified. Options are **Least Frequently Used** (removes tiles based on how often the tile was accessed) or **Least Recently Used** (removes tiles based on date of last access). The optimum configuration is dependent on your data and server usage.

14.1.6 BlobStores

BlobStores allow us to configure how and where GeoWebCache will store its cashed data on a per-layer basis. This page allows us to define the different BlobStores present in the system. BlobStores can be created, modified and removed from here.



General

Identifier

Each BlobStore has a unique identifier.

BlobStore Type

There can be different BlobStore types to use different ways of storage. There is only standard support for File BlobStores. Plugins may add additional types.

Enabled

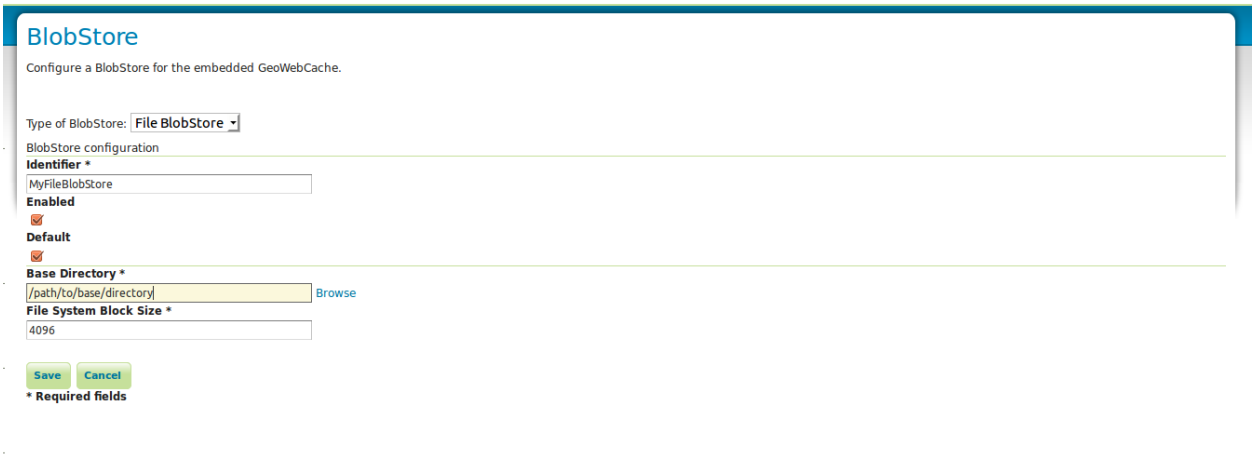
Disabled BlobStores will not be loaded. Disabling a BlobStore will disable cashing for all layers and layer-groups assigned to that BlobStore.

Default

There should always be one default BlobStore, which cannot be removed. The default BlobStore will be used by all layers not assigned to a specific BlobStore. Removing a BlobStore will cause all layers assigned to this BlobStore to use the default BlobStore until specified otherwise.

File BlobStore

These store data on a disk in a specified directory.



Base Directory

The directory where the cashed data is stored.

Disk block size

This setting determines how the tile cache calculates disk usage. The value for this setting should be equivalent to the disk block size of the storage medium where the cache is located. The default block size is **4096 bytes**.

14.2 Using GeoWebCache

Note: For an more in-depth discussion of using GeoWebCache, please see the GeoWebCache documentation.

14.2.1 Direct integration with GeoServer WMS

GeoWebCache can be transparently integrated with the GeoServer WMS, and so requires no special endpoint or custom URL. In this way one can have the simplicity of a standard WMS endpoint with the performance of a tiled client.

Although this direct integration is disabled by default, it can be enabled by going to the *Caching defaults* page in the *Web administration interface*.

When this feature is enabled, GeoServer WMS will cache and retrieve tiles from GeoWebCache (via a GetMap request) only if **all of the following criteria are followed**:

- WMS Direct integration is enabled (you can set this on the *Caching defaults* page)
- `tiled=true` is included in the request
- The request only references a single layer
- Caching is enabled for that layer
- The image requested is of the same height and width as the size saved in the layer configuration
- The requested CRS matches one of the available tile layer gridsets
- The image requested lines up with the existing grid bounds
- A parameter is included for which there is a corresponding Parameter Filter

In addition, when direct integration is enabled, the WMS capabilities document (via a GetCapabilities request) will only return the WMS-C vendor-specific capabilities elements (such as a `<TileSet>` element for each cached layer/CRS/format combination) if `tiled=true` is appended to the GetCapabilities request.

Note: For more information on WMS-C, please see the WMS Tiling Client Recommendation from OSGeo.

Note: GeoWebCache integration is not compatible with the OpenLayers-based *Layer Preview*, as the preview does not usually align with the GeoWebCache layer gridset. This is because the OpenLayers application calculates the `tileorigin` based on the layer's bounding box, which is different from the gridset. It is, possible to create an OpenLayers application that caches tiles; just make sure that the `tileorigin` aligns with the gridset.

Virtual services

When direct WMS integration is enabled, GeoWebCache will properly handle requests to *Virtual Services* (`/geoserver/<workspace>/wms?tiled=true&...`).

Virtual services capabilities documents will contain `<TileSet>` entries only for the layers that belong to that workspace (and global layer groups), and will be referenced by unqualified layer names (no namespace). For example, the layer `topp:states` will be referred to as `<Layers>states</Layers>` instead of `<Layers>topp:states</Layers>`, and GetMap requests to the virtual services endpoint using `LAYERS=states` will properly be handled.

Supported parameter filters

With direct WMS integration, the following parameter filters are supported for GetMap requests:

- `ANGLE`

- BGCOLOR
- BUFFER
- CQL_FILTER
- ELEVATION
- ENV
- FEATUREID
- FEATUREVERSION
- FILTER
- FORMAT_OPTIONS
- MAXFEATURES
- PALETTE
- STARTINDEX
- TIME
- VIEWPARAMS

If a request is made using any of the above parameters, the request will be passed to GeoServer, unless a parameter filter has been set up, in which case GeoWebCache will process the request.

14.2.2 GeoWebCache endpoint URL

When not using direct integration, you can point your client directly to GeoWebCache.

Warning: GeoWebCache is not a true WMS, and so the following is an oversimplification. If you encounter errors, see the *Troubleshooting* page for help.

To direct your client to GeoWebCache (and thus receive cached tiles) you need to change the WMS URL.

If your application requests WMS tiles from GeoServer at this URL:

```
http://example.com/geoserver/wms
```

You can invoke the GeoWebCache WMS instead at this URL:

```
http://example.com/geoserver/gwc/service/wms
```

In other words, add `/gwc/service/wms` in between the path to your GeoServer instance and the WMS call.

This end-point works using either:

- WMS-C: A tileset description is included in the WMS GetCapabilities document instructing clients how to retrieve content as a series of tiles (each retrieved by a GetMap request). This technique supports HTTP caching taking advantage of the browser cache and any caching proxies deployed. This technique requires a client to be created with tile server support.
- full-wms mode: GeoWebCache behaves as normal WMS supported ad-hoc WMS GetMapRequests. Each WMS Request is handled by obtaining the tiles required and stitching the result into a single

image. This technique relies only on internal tile cache, but supports ad-hoc GetMap requests and does not require a client be constructed with tile server support.

To enable this mode add the following in `geowebcache.xml` configuration file:

```
<fullWMS>true</fullWMS>
```

The fullWMS setting only effects the `/gwc/service/wms` endpoint and is not used by direct WMS integration.

As soon as tiles are requested through the `gwc/service/wms` endpoint GeoWebCache automatically starts saving them. The initial requests for each tile will not be accelerated since GeoServer will need to generate the tile and store it from subsequent use. To automate this process of requesting tiles, you can **seed** the cache. See the section on *Seeding and refreshing* for more details.

14.2.3 Disk quota

GeoWebCache has a built-in disk quota feature to prevent disk space from growing unbounded. You can set the maximum size of the cache directory, poll interval, and what policy of tile removal to use when the quota is exceeded. Tiles can be removed based on usage ("Least Frequently Used" or LFU) or timestamp ("Least Recently Used" or LRU).

Disk quotas are turned off by default, but can be configured on the *Disk Quotas* page in the *Web administration interface*.

14.2.4 Integration with external mapping sites

The documentation on the GeoWebCache homepage contains examples for creating applications that integrate with Google Maps, Google Earth, Bing Maps, and more.

14.2.5 Support for custom projections

The version of GeoWebCache that comes embedded in GeoServer automatically configures every layer served in GeoServer with the two most common projections:

- **EPSG:4326** (latitude/longitude)
- **EPSG:900913** (Spherical Mercator, the projection used in Google Maps)

You can also set a custom CRS from any that GeoServer recognizes. See the *Gridsets* page for details.

14.3 Configuration

GeoWebCache is automatically configured for use with GeoServer using the most common options, with no setup required. All communication between GeoServer and GeoWebCache happens by passing messages inside the JVM.

By default, all layers served by GeoServer will be known to GeoWebCache. See the *Tile Layers* page to test the configuration.

Note: Version 2.2.0 of GeoServer introduced changes to the configuration of the integrated GeoWebCache.

14.3.1 Integrated user interface

GeoWebCache has a full integrated web-based configuration. See the *GeoWebCache settings* section in the *Web administration interface*.

14.3.2 Determining tiled layers

In versions of GeoServer prior to 2.2.0, the GeoWebCache integration was done in a such way that every GeoServer layer and layer group was forced to have an associated GeoWebCache tile layer. In addition, every such tile layer was forcedly published in the EPSG:900913 and EPSG:4326 gridsets with PNG and JPEG output formats.

It is possible to selectively turn caching on or off for any layer served through GeoServer. This setting can be configured in the *Tile Layers* section of the *Web administration interface*.

14.3.3 Configuration files

It is possible to configure most aspects of cached layers through the *GeoWebCache settings* section in the *Web administration interface* or the *GeoWebCache REST API*.

GeoWebCache keeps the configuration for each GeoServer tiled layer separately, inside the `<data_dir>/gwc-layers/` directory. There is one XML file for each tile layer. These files contain a different syntax from the `<wmsLayer>` syntax in the standalone version and are *not* meant to be edited by hand. Instead you can configure tile layers on the *Tile Layers* page or through the *GeoWebCache REST API*.

Configuration for the defined gridsets is saved in `<data_dir>/gwc/geowebcache.xml`` so that the integrated GeoWebCache can continue to serve externally-defined tile layers from WMS services outside GeoServer.

If upgrading from a version prior to 2.2.0, a migration process is run which creates a tile layer configuration for all the available layers and layer groups in GeoServer with the old defaults. From that point on, you should configure the tile layers on the *Tile Layers* page.

14.3.4 Changing the cache directory

GeoWebCache will automatically store cached tiles in a `gwc` directory inside your GeoServer data directory. To set a different directory, stop GeoServer (if it is running) and add the following code to your GeoServer `web.xml` file (located in the `WEB-INF` directory):

```
<context-param>
   <param-name>GEOWEBCACHE_CACHE_DIR</param-name>
   <param-value>C:\temp</param-value>
</context-param>
```

Change the path inside `<param-value>` to the desired cache path (such as `C:\temp` or `/tmp`). Restart GeoServer when done.

Note: Make sure GeoServer has write access in this directory.

14.3.5 GeoWebCache with multiple GeoServer instances

For stability reasons, it is not recommended to use the embedded GeoWebCache with multiple GeoServer instances. If you want to configure GeoWebCache as a front-end for multiple instances of GeoServer, we recommend using the standalone GeoWebCache.

14.3.6 GeoServer Data Security

GWC Data Security is an option that can be turned on and turned off through the *Caching defaults* page. By default it is turned off.

When turned on, the embedded GWC will do a data security check before calling GeoWebCache, i.e. verify whether the user actually has access to the layer, and reject the request if this is not the case. In the case of WMS-C requests, there is also limited support for data access limit filters, only with respect to geographic boundaries (all other types of data access limits will be ignored). The embedded GWC will reject requests for which the requested bounding box is (partly) inaccessible. It is only possible to request a tile within a bounding box that is fully accessible. This behaviour is different from the regular WMS, which will filter the data before serving it. However, if the integrated WMS/WMS-C is used, the request will be forwarded back to WMS and give the desired result.

When using the default GeoServer security system, rules cannot combine data security with service security. However, when using a security subsystem it may be possible to make such particular combinations. In this case the WMS-C service inherits all security rules from the regular WMS service; while all other GWC services will get their security from rules associated with the 'GWC' service itself.

14.3.7 Configuring In Memory Caching

GWC In Memory Caching is a new feature which allows to cache GWC tiles in memory reducing their access time. User can also choose to avoid to store the files on the disk if needed. For enabling/disabling these features the user may see the related section on the TileCaching *Caching defaults* page.

Actually there are only two Caching methods:

- Guava Caching
- Hazelcast Caching

Guava Cache

Guava Cache provides a local in-memory cache to use for a single GeoServer instance. For configuring Guava Caching the user must only edit the configuration parameters in the *Caching Defaults* page.

Hazelcast Cache

Hazelcast is an open-source API for distributed structures like clusters. GWC supports this API for creating a distributed in memory cache. At the time of writing, Hazelcast requires the installation of the *gs-gwc-distributed* plugin in the *WEB_INF/lib* directory of your geoserver application. This plugin can be found at the following link.

There are 2 ways for configuring distributed caching in GWC:

- Using an XML file called *hazelcast.xml*. This file must be located in a directory indicated by the JVM parameter **hazelcast.config.dir**
- Directly, by configuring a bean inside the GeoServer application context

Both the 2 configurations should define the following parameters:

1. The Hazelcast configuration requires a Map object with name *CacheProviderMap*
2. Map eviction policy must be *LRU* or *LFU*
3. Map configuration must have a fixed size defined in Mb
4. Map configuration must have **USED_HEAP_SIZE** as *MaxSizePolicy*

Warning: Be careful that all the cluster instances have the same configuration, because different configurations may result in incorrect Hazelcast behaviour.

Warning: In order to avoid missing tiles, the cluster instances should access the same data.

Configuration with *hazelcast.xml*

Here can be found an example file:

```
<hazelcast xsi:schemaLocation="http://www.hazelcast.com/schema/config␣
↪hazelcast-config-2.3.xsd"
                   xmlns="http://www.hazelcast.com/schema/config"
                   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <group>
        <name>cacheCluster</name>
        <password>geoserverCache</password>
  </group>

  <network>
        <!--
                Typical usage: multicast enabled with port auto-increment␣
↪enabled
                or tcp-ip enabled with port auto-increment disabled. Note␣
↪that you
                must choose between multicast and tcp-ip. Another option␣
↪could be
                aws, but will not be described here.

        -->
        <port auto-increment="false">5701</port>
                <join>
                         <multicast enabled="false">
                                <multicast-group>224.2.2.3</multicast-group>
                                <multicast-port>54327</multicast-port>
                        </multicast>
                        <tcp-ip enabled="true">
                                <interface>192.168.1.32</interface>
                                <interface>192.168.1.110</interface>
                        </tcp-ip>
                </join>
  </network>

  <map name="CacheProviderMap">
                <eviction-policy>LRU</eviction-policy>
```

```
                <max-size policy="USED_HEAP_SIZE">16</max-size>
  </map>

</hazelcast>
```

Configuration with ApplicationContext

For configuring caching directly from the GeoServer application context, the user must edit the file *geowebcache-distributed.xml* inside the *gs-gwc* jar file. More informations about using Spring with Hazelcast can be found in the Hazelcast related documentation. The modified application context is presented below:

```
<hz:hazelcast id="instance1">
        <hz:config>
                <hz:group name="dev" password="password" />
                <hz:network port="5701" port-auto-increment="true">
                        <hz:join>
                                <hz:multicast enabled="true" multicast-group=
↪"224.2.2.3"
                                        multicast-port="54327" />
                        <hz:tcp-ip enabled="false">
                          <hz:members>10.10.1.2, 10.10.1.3</hz:members>
                        </hz:tcp-ip>
                        </hz:join>
                </hz:network>
                <hz:map name="CacheProviderMap" max-size="16" eviction-
↪policy="LRU"
                        max-size-policy="USED_HEAP_SIZE" />
        </hz:config>
</hz:hazelcast>

<bean id="HazelCastLoader1"
        class="org.geowebcache.storage.blobstore.memory.distributed.
↪HazelcastLoader">
        <property name="instance" ref="instance1" />
</bean>

<bean id="HazelCastCacheProvider1"
        class="org.geowebcache.storage.blobstore.memory.distributed.
↪HazelcastCacheProvider">
        <constructor-arg ref="HazelCastLoader1" />
</bean>
```

Optional configuration parameters

In this section are described other available configuration parameters to configure:

- Cache expiration time:

```
<map name="CacheProviderMap">
...

        <time-to-live-seconds>0</time-to-live-seconds>
        <max-idle-seconds>0</max-idle-seconds>
```

```
</map>
```

Where *time-to-live-seconds* indicates how many seconds an entry can stay in cache and *max-idle-seconds* indicates how many seconds an entry may be not accessed before being evicted.

- Near Cache.

```
<map name="CacheProviderMap">
...
<near-cache>
  <!--
        Same configuration parameters of the Hazelcast Map. Note␣
↪that size indicates the maximum number of
        entries in the near cache. A value of Integer.MAX_VALUE␣
↪indicates no limit on the maximum
        size.
  -->
  <max-size>5000</max-size>
  <time-to-live-seconds>0</time-to-live-seconds>
  <max-idle-seconds>60</max-idle-seconds>
  <eviction-policy>LRU</eviction-policy>

  <!--
        Indicates if a cached entry can be evicted if the same␣
↪value is modified in the Hazelcast Map. Default is true.
  -->
  <invalidate-on-change>true</invalidate-on-change>

  <!--
        Indicates if local entries must be cached. Default is␣
↪false.
  -->
  <cache-local-entries>false</cache-local-entries>
</near-cache>

</map>
```

Near Cache is a local cache for each cluster instance which is used for caching entries in the other cluster instances. This behaviour avoids requesting those entries each time by executing a remote call. This feature could be helpful in order to improve Hazelcast Cache performance.

Note: A value of *max-size* bigger or equal to Integer.MAX_VALUE cannot be used in order to avoid an uncontrollable growth of the cache size.

14.4 Seeding and refreshing

The primary benefit to GeoWebCache is that it allows for the acceleration of normal WMS tile request processing by eliminating the need for the tiles to be regenerated for every request. This page discusses tile generation.

You can configure seeding processes via the *Web administration interface*. See the *Tile Layers* page for more information.

14.4.1 Generating tiles

There are two ways for tiles to be generated by GeoWebCache. The first way for tiles to be generated is during **normal map viewing**. In this case, tiles are cached only when they are requested from a client, either through map browsing (such as in OpenLayers) or through manual WMS tile requests. The first time a map view is requested it will be roughly at the same speed as a standard GeoServer WMS request. The second and subsequent map viewings will be greatly accelerated as those tiles will have already been generated. The main advantage to this method is that it requires no preprocessing, and that only the data that has been requested will be cached, thus potentially saving disk space as well. The disadvantage to this method is that map viewing will be only intermittently accelerated, reducing the quality of user experience.

The other way for tiles to be generated is by **seeding**. Seeding is the process where map tiles are generated and cached internally from GeoWebCache. When processed in advance, the user experience is greatly enhanced, as the user never has to wait for tiles to be generated. The disadvantage to this process is that seeding can be a very time- and disk-consuming process.

In practice, a combination of both methods are usually used, with certain zoom levels (or certain areas of zoom levels) seeded, and the less-likely-viewed tiles are left uncached.

14.5 HTTP Response Headers

The GeoWebCache integrated with GeoServer employs special information stored in the header of responses. These headers are available either with direct calls to the *GeoWebCache endpoint* or with *direct WMS integration*.

14.5.1 Custom response headers

GeoWebCache returns both standard and custom HTTP response headers when serving a tile request. This aids in the debugging process, as well as adhering to an HTTP 1.1 transfer control mechanism.

The response headers can be determined via a utility such as cURL.

Example

Note: For all cURL commands below, make sure to replace `>/dev/null` with `>nul` if you are running on Windows.

This is a sample request and response using cURL:

```
curl -v "http://localhost:8080/geoserver/gwc/service/wms?LAYERS=sde%3Abmworld&
↪FORMAT=image%2Fpng&SERVICE=WMS&VERSION=1.1.1&REQUEST=GetMap&STYLES=&SRS=EPSG%3A4326&
↪BBOX=-180,-38,-52,90&WIDTH=256&HEIGHT=256&tiled=true" > /dev/null
```

```
< HTTP/1.1 200 OK
< geowebcache-tile-index: [0, 1, 2]
< geowebcache-cache-result: HIT
< geowebcache-tile-index: [0, 1, 2]
< geowebcache-tile-bounds: -180.0,-38.0,-52.0,90.0
< geowebcache-gridset: GlobalCRS84Pixel
< geowebcache-crs: EPSG:4326
< Content-Type: image/png
```

```
< Content-Length: 102860
< Server: Jetty(6.1.8)
```

From this, one can learn that the tile was found in the cache (`HIT`), the requested tile was from the gridset called `GlobalCRS84Pixel` and had a CRS of `EPSG:4326`.

List of custom response headers

The following is the full list of custom response headers. Whenever GeoWebCache serves a tile request, it will write some or all of the following custom headers on the HTTP response.

Response Header	Description
`geowebcache-cache-result`	Shows whether the GeoWebCache WMS was used. Options are: • `HIT`: Tile requested was found on the cache • `MISS`: Tile was not found on the cache but was acquired from the layer's data source • `WMS`: Request was proxied directly to the origin WMS (for example, for GetFeatureInfo requests) • `OTHER`: Response was the default white/transparent tile or an error occurred
`geowebcache-tile-index`	Contains the three-dimensional tile index in x,y,z order of the returned tile image in the corresponding grid space (e.g. `[1, 0, 0]`)
`geowebcache-tile-bounds`	Bounds of the returned tile in the corresponding coordinate reference system (e.g. `-180,-90,0,90`)
`geowebcache-gridset`	Name of the gridset the tile belongs to (see *Gridsets* for more information)
`geowebcache-crs`	Coordinate reference system code of the matching gridset (e.g. `EPSG:900913`, `EPSG:4326`, etc).

14.5.2 Last-Modified and If-Modified-Since

Well behaved HTTP 1.1 clients and server applications can make use of `Last-Modified` and `If-Modified-Since` HTTP control mechanisms to know when locally cached content is up to date, eliminating the need to download the same content again. This can result in considerable bandwidth savings. (See HTTP 1.1 RFC 2616, sections 14.29 and 14.25, for more information on these mechanisms.)

GeoWebCache will write a `Last-Modified` HTTP response header when serving a tile image. The date is written as an RFC-1123 `HTTP-Date`:

```
Last-Modified: Wed, 15 Nov 1995 04:58:08 GMT
```

Clients connecting to GeoWebCache can create a "conditional GET" request with the `If-Modified-Since` request header. If the tile wasn't modified after the date specified in the `Last-Modified` response header, GeoWebCache will return a `304` status code indicating that the resource was available and not modified.

Example

A query for a specific tile returns the `Last-Modified` response header:

```
curl -v "http://localhost:8080/geoserver/gwc/service/wms?LAYERS=img%20states&
↪FORMAT=image%2Fpng&SERVICE=WMS&VERSION=1.1.1&REQUEST=GetMap&STYLES=&
↪EXCEPTIONS=application%2Fvnd.ogc.se_inimage&SRS=EPSG%3A4326&BBOX=-135,45,-90,90&
↪WIDTH=256&HEIGHT=256" >/dev/null
```

```
> Host: localhost:8080
> Accept: */*
>
< HTTP/1.1 200 OK
...
< Last-Modified: Wed, 25 Jul 2012 00:42:00 GMT
< Content-Type: image/png
< Content-Length: 31192
```

This request has the `If-Modified-Since` header set to one second after what was returned by `Last-Modified`:

```
curl --header "If-Modified-Since: Wed, 25 Jul 2012 00:42:01 GMT" -v "http://
↪localhost:8080/geoserver/gwc/service/wms?LAYERS=img%20states&FORMAT=image%2Fpng&
↪SERVICE=WMS&VERSION=1.1.1&REQUEST=GetMap&STYLES=&EXCEPTIONS=application%2Fvnd.ogc.
↪se_inimage&SRS=EPSG%3A4326&BBOX=-135,45,-90,90&WIDTH=256&HEIGHT=256" >/dev/null
```

```
> Host: localhost:8080
> Accept: */*
> If-Modified-Since: Wed, 25 Jul 2012 00:42:01 GMT
>
< HTTP/1.1 304 Not Modified
< Last-Modified: Wed, 25 Jul 2012 00:42:00 GMT
< Content-Type: image/png
< Content-Length: 31192
```

The response code is `304`. As the file hasn't been modified since the time specified in the request, no content is actually transferred. The client is informed that its copy of the tile is up to date.

However, if you were to set the `If-Modified-Since` header to *before* the time stored in `Last-Modified`, you will instead receive a `200` status code and the tile will be downloaded.

This example sets the `If-Modified-Since` header to one second before what was returned by `Last-Modified`:

```
curl --header "If-Modified-Since: Wed, 25 Jul 2012 00:41:59 GMT" -v "http://
↪localhost:8080/geoserver/gwc/service/wms?LAYERS=img%20states&FORMAT=image%2Fpng&
↪SERVICE=WMS&VERSION=1.1.1&REQUEST=GetMap&STYLES=&EXCEPTIONS=application%2Fvnd.ogc.
↪se_inimage&SRS=EPSG%3A4326&BBOX=-135,45,-90,90&WIDTH=256&HEIGHT=256" >/dev/null
```

```
> Host: localhost:8080
> Accept: */*
> If-Modified-Since: Wed, 25 Jul 2012 00:41:59 GMT
>
< HTTP/1.1 200 OK
...
< Last-Modified: Wed, 25 Jul 2012 00:42:00 GMT
< Content-Type: image/png
< Content-Length: 31192
```

14.6 GeoWebCache REST API

This section discusses the GeoWebCache REST API, an interface for working programmatically with the integrated GeoWebCache without the need for a GUI.

The GeoWebCache REST endpoint when integrated with GeoServer is available at:

```
<GEOSERVER_HOME>/gwc/rest/
```

For example:

```
http://example.com:8080/geoserver/gwc/rest/
```

14.6.1 Managing Layers

The GeoWebCache REST API provides a RESTful interface through which users can add, modify, or remove cached layers.

Note: JSON is not recommended for managing layers as the JSON library has a number of issues with multi-valued properties such as "parameterFilters".

Layer list

URL: `/gwc/rest/layers.xml`

Method	Action	Return Code	Formats
GET	Return the list of available layers	200	XML
POST		400	
PUT		400	
DELETE		400	

The following example will request a full list of layers:

```
curl -u admin:geoserver "http://localhost:8080/geoserver/gwc/rest/layers"
```

```
<layers>
  <layer>
    <name>img states</name>
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate" href="http://
↪localhost:8080/geoserver/gwc/rest/layers/img+states.xml" type="text/xml"/>
  </layer>
  <layer>
    <name>raster test layer</name>
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate" href="http://
↪localhost:8080/geoserver/gwc/rest/layers/raster+test+layer.xml" type="text/xml"/>
  </layer>
  <layer>
    <name>topp:states</name>
```

```
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate" href="http://
↪localhost:8080/geoserver/gwc/rest/layers/topp%3Astates.xml" type="text/xml"/>
  </layer>
 </layers>
```

Layer Operations

URL: `/gwc/rest/layers/<layer>.xml`

Note: JSON is not recommended for managing layers as the JSON library has a number of issues with multi-valued properties such as "parameterFilters".

Method	Action	Return Code	Formats
GET	Return the XML representation of the layer	200	XML
POST	Modify the definition/configuration of the layer	200	XML
PUT	Add a new layer	200	XML
DELETE	Delete the layer	200	

Note: There are two different representations for cached layers, depending on whether the tile layer is created from the GeoServer WMS layer or layer group (`GeoServerLayer`), or is configured in `geowebcache.xml` as a regular GWC layer (`wmsLayer`). A GeoServer layer is referred to as a `GeoServerLayer` and contains no image data source information such as origin WMS URL.

Representations:

- GeoWebCache (`wmsLayer`) `XML minimal`
- GeoWebCache (`wmsLayer`) `XML`
- GeoServer (`GeoServerLayer`) `XML minimal`
- GeoServer (`GeoServerLayer`) `XML`

The examples below use the cURL tool, though the examples apply to any HTTP-capable tool or library.

Adding a GeoWebCache layer

The following example will add a new layer to GeoWebCache:

```
curl -v -u admin:geoserver -XPUT -H "Content-type: text/xml" -d @layer.xml  "http://
↪localhost:8080/geoserver/gwc/rest/layers/newlayer.xml"
```

The `layer.xml` file is defined as the following:

```
<wmsLayer>
  <name>newlayer</name>
  <mimeFormats>
    <string>image/png</string>
  </mimeFormats>
  <gridSubsets>
    <gridSubset>
```

```
      <gridSetName>EPSG:900913</gridSetName>
    </gridSubset>
  </gridSubsets>
  <wmsUrl>
    <string>http://localhost:8080/geoserver/wms</string>
  </wmsUrl>
  <wmsLayers>topp:states</wmsLayers>
</wmsLayer>
```

Note: The addressed resource (`newlayer` in this example) must match the name of the layer in the XML representation.

Adding a GeoServer layer

The following example will add a new layer to both GeoServer and GeoWebCache:

```
curl -v -u admin:geoserver -XPUT -H "Content-type: text/xml" -d @poi.xml  "http://
↪localhost:8080/geoserver/gwc/rest/layers/tiger:poi.xml"
```

The `poi.xml` file is defined as the following:

```
<GeoServerLayer>
  <id>LayerInfoImpl--570ae188:124761b8d78:-7fd0</id>
  <enabled>true</enabled>
  <name>tiger:poi</name>
  <mimeFormats>
    <string>image/png8</string>
  </mimeFormats>
  <gridSubsets>
    <gridSubset>
      <gridSetName>GoogleCRS84Quad</gridSetName>
      <zoomStart>0</zoomStart>
      <zoomStop>14</zoomStop>
      <minCachedLevel>1</minCachedLevel>
      <maxCachedLevel>9</maxCachedLevel>
    </gridSubset>
  </gridSubsets>
  <metaWidthHeight>
    <int>4</int>
    <int>4</int>
  </metaWidthHeight>
  <gutter>50</gutter>
  <autoCacheStyles>true</autoCacheStyles>
</GeoServerLayer>
```

Note: The addressed resource (`tiger:poi` in this example) must match the name of the layer in the XML representation, as well as the name of an *existing* GeoServer layer or layer group.

Modifying a layer

This example modifies the layer definition via the `layer.xml` file. The request adds a parameter filter and a grid subset to the existing `tiger:poi` tile layer:

```
<GeoServerLayer>
 <enabled>true</enabled>
 <name>tiger:poi</name>
 <mimeFormats>
   <string>image/png8</string>
 </mimeFormats>
 <gridSubsets>
   <gridSubset>
     <gridSetName>GoogleCRS84Quad</gridSetName>
     <zoomStart>0</zoomStart>
     <zoomStop>14</zoomStop>
     <minCachedLevel>1</minCachedLevel>
     <maxCachedLevel>9</maxCachedLevel>
   </gridSubset>
   <gridSubset>
     <gridSetName>EPSG:900913</gridSetName>
     <extent>
       <coords>
         <double>-8238959.403861314</double>
         <double>4969300.121476209</double>
         <double>-8237812.689219721</double>
         <double>4971112.167757057</double>
       </coords>
     </extent>
   </gridSubset>
 </gridSubsets>
 <metaWidthHeight>
   <int>4</int>
   <int>4</int>
 </metaWidthHeight>
 <parameterFilters>
   <floatParameterFilter>
     <key>ELEVATION</key>
     <defaultValue>0.0</defaultValue>
     <values>
       <float>0.0</float>
       <float>1.0</float>
       <float>2.0</float>
       <float>3.0</float>
       <float>4.0</float>
     </values>
     <threshold>1.0E-3</threshold>
   </floatParameterFilter>
 </parameterFilters>
 <gutter>50</gutter>
 <autoCacheStyles>true</autoCacheStyles>
</GeoServerLayer>
```

Instead of PUT, use the HTTP POST method instead:

```
curl -v -u admin:geoserver -XPOST -H "Content-type: text/xml" -d @poi.xml  "http://
↪localhost:8080/geoserver/gwc/rest/layers/tiger:poi.xml"
```

Deleting a layer

Deleting a GeoWebCache tile layer deletes the layer configuration *as well as the layer's disk cache*. No tile images will remain in the cache directory after deleting a tile layer.

To delete a layer, use the HTTP DELETE method against the layer resource:

```
curl -v -u admin:geoserver -XDELETE "http://localhost:8080/geoserver/gwc/rest/layers/
↪newlayer.xml"
```

Note: If trying to delete a tile layer that is an integrated `GeoServerLayer`, only the GeoWebCache layer definition will be deleted; the GeoServer definition is left untouched. To delete a layer in GeoServer, use the GeoServer *REST* to manipulate GeoServer resources.

On the other hand, deleting a GeoServer layer via the GeoServer REST API *will* automatically delete the associated tile layer.

14.6.2 Seeding and Truncating

The GeoWebCache REST API provides a RESTful interface through which users can add or remove tiles from the cache on a per-layer basis.

Operations

URL: `/gwc/rest/seed/<layer>.<format>`

Method	Action	Return Code	Formats
GET	Return the status of the seeding threads	200	JSON
POST	Issue a seed or truncate task request	200	XML, JSON
PUT		405	
DELETE		405	

Representations:

- `XML`
- `JSON`

The examples below use the cURL tool, though the examples apply to any HTTP-capable tool or library.

Seeding

The following XML request initiates a seeding task:

```
curl -v -u admin:geoserver -XPOST -H "Content-type: text/xml" -d '<seedRequest><name>
↪nurc:Arc_Sample</name><srs><number>4326</number></srs><zoomStart>1</zoomStart>
↪<zoomStop>12</zoomStop><format>image/png</format><type>truncate</type><threadCount>2
↪</threadCount></seedRequest>'  "http://localhost:8080/geoserver/gwc/rest/seed/
↪nurc:Arc_Sample.xml"
```

```
* About to connect() to localhost port 8080 (#0)
*   Trying 127.0.0.1... connected
* Connected to localhost (127.0.0.1) port 8080 (#0)
* Server auth using Basic with user 'admin'
> POST /geoserver/gwc/rest/seed/nurc:Arc_Sample.xml HTTP/1.1
> Authorization: Basic YWRtaW46Z2Vvc2VydmVy
> User-Agent: curl/7.21.3 (x86_64-pc-linux-gnu) libcurl/7.21.3 OpenSSL/0.9.8o zlib/1.
↪2.3.4 libidn/1.18
> Host: localhost:8080
> Accept: */*
> Content-type: text/xml
> Content-Length: 209
>
< HTTP/1.1 200 OK
```

The following is a more complete XML fragment for a seed request, including parameter filters:

```
<?xml version="1.0" encoding="UTF-8"?>
<seedRequest>
  <name>topp:states</name>
  <bounds>
    <coords>
      <double>-2495667.977678598</double>
      <double>-2223677.196231552</double>
      <double>3291070.6104286816</double>
      <double>959189.3312465074</double>
    </coords>
  </bounds>

  <!-- These are listed on http://localhost:8080/geoserver/gwc/demo -->
  <gridSetId>EPSG:2163</gridSetId>
  <zoomStart>0</zoomStart>
  <zoomStop>2</zoomStop>
  <format>image/png</format>

  <!-- type can be seed, reseed, or truncate -->
  <type>truncate</type>

  <!-- Number of seeding threads to run in parallel.
       If type == truncate only one thread will be used
       regardless of this parameter -->
  <threadCount>1</threadCount>
  <!-- Parameter filters -->
  <parameters>
    <entry>
      <string>STYLES</string>
      <string>pophatch</string>
    </entry>
    <entry>
      <string>CQL_FILTER</string>
      <string>TOTPOP > 10000</string>
    </entry>
  </parameters>
</seedRequest>
```

Truncating

The following XML request initiates a truncating task:

```
curl -v -u admin:geoserver -XPOST -H "Content-type: application/json" -d "{
↪'seedRequest':{'name':'topp:states','bounds':{'coords':{ 'double':['-124.0','22.0',
↪'66.0','72.0']}},'srs':{'number':4326},'zoomStart':1,'zoomStop':12,'format':'image\/
↪png','type':'truncate','threadCount':4}}}"  "http://localhost:8080/geoserver/gwc/
↪rest/seed/nurc:Arc_Sample.json"
```

```
* About to connect() to localhost port 8080 (#0)
*   Trying 127.0.0.1... connected
* Connected to localhost (127.0.0.1) port 8080 (#0)
* Server auth using Basic with user 'admin'
> POST /geoserver/gwc/rest/seed/nurc:Arc_Sample.json HTTP/1.1
> Authorization: Basic YWRtaW46Z2Vvc2VydmVy
> User-Agent: curl/7.21.3 (x86_64-pc-linux-gnu) libcurl/7.21.3 OpenSSL/0.9.8o zlib/1.
↪2.3.4 libidn/1.18
> Host: localhost:8080
> Accept: */*
> Content-type: application/json
> Content-Length: 205
>
< HTTP/1.1 200 OK
< Date: Fri, 14 Oct 2011 22:09:21 GMT
< Server: Noelios-Restlet-Engine/1.0..8
< Transfer-Encoding: chunked
<
* Connection #0 to host localhost left intact
* Closing connection #0
```

Querying running tasks

URL: `/gwc/rest/seed[/<layer>].json`

Method	Action	Return Code	Formats
GET	Get the global or per layer state of running and pending tasks	200	JSON
POST		405	
PUT		405	
DELETE		405	

Getting current state of the seeding threads

Sending a GET request to the `/gwc/rest/seed.json` resource returns a list of pending (scheduled) and running tasks for all the layers.

Sending a GET request to the `/gwc/rest/seed/<layer name>.json` resource returns a list of pending (scheduled) and running tasks for that specific layer.

The returned content is a JSON array of the form:

```
{"long-array-array":[[<long>,<long>,<long>,<long>,<long>],...]}
```

If there are no pending or running tasks, the returned array is empty:

```
{"long-array-array":[]}
```

The returned array of arrays contains one array per seeding/truncating task. The meaning of each long value in each thread array is:

```
[tiles processed, total # of tiles to process, # of remaining tiles, Task ID, Task␣
↪status]
```

The returned `Task Status` value will be one of the following:

```
-1 = ABORTED
 0 = PENDING
 1 = RUNNING
 2 = DONE
```

The example below returns the current state of tasks for the `topp:states` layer:

```
curl -u <user>:<password> -v -XGET http://localhost:8080/geoserver/gwc/rest/seed/
↪topp:states.json
```

```
{"long-array-array":[[17888,44739250,18319,1,1],[17744,44739250,18468,2,1],[16608,
↪44739250,19733,3,0],[0,1000,1000,4,0]]}
```

In the above response, tasks `1` and `2` for the `topp:states` layer are running, and tasks `3` and `4` are in a pending state waiting for an available thread.

The example below returns a list of tasks for all the layers.

```
curl -u <user>:<password> -XGET http://localhost:8080/geoserver/gwc/rest/seed.json
```

```
{"long-array-array":[[2240,327426,1564,2,1],[2368,327426,1477,3,1],[2272,327426,1541,
↪4,1],[2176,327426,1611,5,1],[1056,15954794690,79320691,6,1],[1088,15954794690,
↪76987729,7,1],[1040,15954794690,80541010,8,1],[1104,15954794690,75871965,9,1]]}
```

Terminating running tasks

URL: `/gwc/rest/seed[/<layer>]`

Method	Action	Return Code	Formats
GET		405	
POST	Issue a kill running and/or pending tasks request	200	
PUT		405	
DELETE		405	

A POST request to the `/gwc/rest/seed` resource terminates pending and/or running tasks for **all layers**. A POST request to the `/gwc/rest/seed/<layername>` resource terminates pending and/or running tasks for a specific layer.

It is possible to terminate individual or all pending and/or running tasks. Use the parameter `kill_all` with one of the following values: `running`, `pending`, or `all`.

Note: For backward compatibility, the `kill_all` parameter value `1` is also accepted and is equivalent to `running`.

The following request terminates all running seed and truncate tasks.

```
curl -v -u admin:geoserver -d "kill_all=all"  "http://localhost:8080/geoserver/gwc/
↪rest/seed"
```

```
* About to connect() to localhost port 8080 (#0)
*   Trying 127.0.0.1... connected
< HTTP/1.1 200 OK
< Date: Fri, 14 Oct 2011 22:23:04 GMT
< Server: Noelios-Restlet-Engine/1.0..8
< Content-Type: text/html; charset=ISO-8859-1
< Content-Length: 426
<
<html>
...
* Connection #0 to host localhost left intact
* Closing connection #0
```

14.6.3 Disk Quota

The GeoWebCache REST API provides a RESTful interface through which users can configure the disk usage limits and expiration policies for a GeoWebCache instance.

Operations

URL: `/gwc/rest/diskquota.<format>`

Method	Action	Return Code	Formats
GET	Return the global disk quota configuration	200	XML, JSON
POST		405	
PUT	Modify the global disk quota configuration	200	XML, JSON
DELETE		405	

Representations:

- XML
- JSON

The examples below use the cURL tool, though the examples apply to any HTTP-capable tool or library.

Retrieving the current configuration

The following returns the current disk quota configuration in **XML** format:

```
curl -u admin:geoserver -v -XGET http://localhost:8080/geoserver/gwc/rest/diskquota.
↪xml
```

```
< HTTP/1.1 200 OK
< Date: Mon, 21 Mar 2011 13:50:49 GMT
< Server: Noelios-Restlet-Engine/1.0..8
< Content-Type: text/xml; charset=ISO-8859-1
< Content-Length: 422
```

```
<
<gwcQuotaConfiguration>
  <enabled>true</enabled>
  <diskBlockSize>2048</diskBlockSize>
  <cacheCleanUpFrequency>5</cacheCleanUpFrequency>
  <cacheCleanUpUnits>SECONDS</cacheCleanUpUnits>
  <maxConcurrentCleanUps>5</maxConcurrentCleanUps>
  <globalExpirationPolicyName>LRU</globalExpirationPolicyName>
  <globalQuota>
    <value>100</value>
    <units>MiB</units>
  </globalQuota>
  <layerQuotas/>
</gwcQuotaConfiguration>
```

The following returns the current disk quota configuration in **JSON** format:

```
curl -u admin:geoserver -v -XGET http://localhost:8080/geoserver/gwc/rest/diskquota.
↪json
```

```
< HTTP/1.1 200 OK
< Date: Mon, 21 Mar 2011 13:53:42 GMT
< Server: Noelios-Restlet-Engine/1.0..8
< Content-Type: application/json; charset=ISO-8859-1
< Content-Length: 241
<
* Connection #0 to host localhost left intact
* Closing connection #0
{"gwcQuotaConfiguration":{"diskBlockSize":2048,"enabled":true,"maxConcurrentCleanUps
↪":5,"cacheCleanUpFrequency":5,"globalExpirationPolicyName":"LRU","globalQuota":{
↪"value":"100","units":"MiB"},"cacheCleanUpUnits":"SECONDS"}}
```

Changing configuration

Note: The request body for PUT should contain only the desired properties to be modified. For example, the following will only change the maxConcurrentCleanups property in XML format:

```
<gwcQuotaConfiguration><maxConcurrentCleanUps>2</maxConcurrentCleanUps></
↪gwcQuotaConfiguration>
```

The following will only change the diskBlockSize, enabled, and globalQuota properties in JSON format:

```
{"gwcQuotaConfiguration":{"diskBlockSize":2048,"enabled":true,"globalQuota":{"value":
↪"100","units":"MiB"}}
```

The following XML example successfully enables the quota and sets the globalQuota size:

```
curl -v -u admin:geoserver "http://localhost:8090/geoserver/gwc/rest/diskquota.xml" -
↪X PUT -d "<gwcQuotaConfiguration><enabled>true</enabled><globalQuota><value>100</
↪value><units>GiB</units></globalQuota></gwcQuotaConfiguration>"
```

```
< HTTP/1.1 200 OK
< Date: Fri, 18 Mar 2011 20:59:31 GMT
```

```
< Server: Noelios-Restlet-Engine/1.0..8
< Content-Type: text/xml; charset=ISO-8859-1
< Content-Length: 422
<
<gwcQuotaConfiguration>
  <enabled>true</enabled>
  <diskBlockSize>2048</diskBlockSize>
  <cacheCleanUpFrequency>5</cacheCleanUpFrequency>
  <cacheCleanUpUnits>SECONDS</cacheCleanUpUnits>
  <maxConcurrentCleanUps>5</maxConcurrentCleanUps>
  <globalExpirationPolicyName>LFU</globalExpirationPolicyName>
  <globalQuota>
    <value>100</value>
    <units>GiB</units>
  </globalQuota>
  <layerQuotas/>
</gwcQuotaConfiguration>
```

The following JSON example changes the globalQuote and expirationPolicyName parameters:

```
curl -v -u admin:geoserver "http://localhost:8090/geoserver/gwc/rest/diskquota.json" -
↪X PUT -d "{"gwcQuotaConfiguration":{"globalQuota":{"value":"100","units":"MiB"},
↪"globalExpirationPolicyName":"LRU"}}"
```

```
< HTTP/1.1 200 OK
< Date: Fri, 18 Mar 2011 21:02:20 GMT
< Server: Noelios-Restlet-Engine/1.0..8
< Content-Type: application/json; charset=ISO-8859-1
< Content-Length: 241
<
* Connection #0 to host localhost left intact
* Closing connection #0
{"gwcQuotaConfiguration":{"diskBlockSize":2048,"enabled":true,"maxConcurrentCleanUps
↪":5,"cacheCleanUpFrequency":5,"globalExpirationPolicyName":"LRU","globalQuota":{
↪"value":"100","units":"MiB"},"cacheCleanUpUnits":"SECONDS","layerQuotas":[]}}
```

The following *invalid* XML example has an invalid parameter (maxConcurrentCleanUps must be > 0). It returns a 400 response code and contains an error message as plain text:

```
curl -v -u admin:geoserver "http://localhost:8090/geoserver/gwc/rest/diskquota.xml" -
↪X PUT -d "<gwcQuotaConfiguration><maxConcurrentCleanUps>-1</maxConcurrentCleanUps></
↪gwcQuotaConfiguration>"
```

```
< HTTP/1.1 400 Bad Request
< Date: Fri, 18 Mar 2011 20:53:26 GMT
< Server: Noelios-Restlet-Engine/1.0..8
< Content-Type: text/plain; charset=ISO-8859-1
< Content-Length: 53
<
* Connection #0 to host localhost left intact
* Closing connection #0
maxConcurrentCleanUps shall be a positive integer: -1
```

The following *invalid* JSON example uses an unknown unit of measure (ZZiB). It returns a 400 response code and contains an error message as plain text:

```
curl -v -u admin:geoserver "http://localhost:8090/geoserver/gwc/rest/diskquota.json" -
↪X PUT -d "{"gwcQuotaConfiguration":{"globalQuota":{"value":"100","units":"ZZiB"}}}"
```

```
< HTTP/1.1 400 Bad Request
< Date: Fri, 18 Mar 2011 20:56:23 GMT
< Server: Noelios-Restlet-Engine/1.0..8
< Content-Type: text/plain; charset=ISO-8859-1
< Content-Length: 601
<
No enum const class org.geowebcache.diskquota.storage.StorageUnit.ZZiB : No enum␣
↪const class org.geowebcache.diskquota.storage.StorageUnit.ZZiB
---- Debugging information ----
message             : No enum const class org.geowebcache.diskquota.storage.
↪StorageUnit.ZZiB
cause-exception     : java.lang.IllegalArgumentException
cause-message       : No enum const class org.geowebcache.diskquota.storage.
↪StorageUnit.ZZiB
class               : org.geowebcache.diskquota.DiskQuotaConfig
required-type       : org.geowebcache.diskquota.storage.Quota
line number         : -1
* Connection #0 to host localhost left intact
* Closing connection #0
```

14.7 Troubleshooting

This section will discuss some common issues with the integrated GeoWebCache and their solutions.

14.7.1 Grid misalignment

Sometimes errors will occur when requesting data from GeoWebCache endpoints. The error displayed might say that the "resolution is not supported" or the "bounds do not align." This is due to the client making WMS requests that do not align with the grid of tiles that GeoWebCache has created, such as differing map bounds or layer bounds, or an unsupported resolution. If you are using OpenLayers as a client, looking at the source code of the included demos may provide more clues to matching up the grid.

An alternative workaround is to enable direct WMS integration with the GeoServer WMS. You can set this on the *Caching defaults* page.

14.7.2 Direct WMS integration

Direct integration allows WMS requests served through GeoServer to be cached as if they were received and processed by GeoWebCache. With Direct WMS Integration, a request may either be handled by the GeoServer WMS or GeoWebCache WMS.

Sometimes requests that should go to GeoWebCache will instead be passed through to GeoServer, resulting in no tiles saved. That said, it is possible to determine why a request was not handled by GeoWebCache when intended. This is done by using the command-line utility cURL and inspecting the response headers.

First, obtain a sample request. This can easily be done by going to the Layer Preview for a given layer, setting the *Tiled* parameter to *Tiled*, then right-clicking on an area of the map and copy the full path to the image location. If done correctly, the result will be a GET request that looks something like this:

```
http://localhost:8090/geoserver/nurc/wms?LAYERS=nurc%3AArc_Sample&STYLES=&FORMAT=image
↪%2Fjpeg&TILED=true&TILESORIGIN=-180%2C-90&SERVICE=WMS&VERSION=1.1.1&REQUEST=GetMap&
↪SRS=EPSG%3A4326&BBOX=-45,-45,0,0&WIDTH=256&HEIGHT=256
```

You can then paste this URL into a curl request:

```
curl -v "URL"
```

For example:

```
curl -v "http://localhost:8090/geoserver/nurc/wms?LAYERS=nurc%3AArc_Sample&STYLES=&
↪FORMAT=image%2Fjpeg&TILED=true&TILESORIGIN=-180%2C-90&SERVICE=WMS&VERSION=1.1.1&
↪REQUEST=GetMap&SRS=EPSG%3A4326&BBOX=-45,-45,0,0&WIDTH=256&HEIGHT=256"
```

Note: To omit the raw image output to the terminal, pipe the output to your system's null. On Linux / OS X, append `> /dev/null` to these requests, and on Windows, append `> nul`.

If the request doesn't go through GeoWebCache's WMS, a reason will be given in a custom response header. Look for the following response headers:

- `geowebcache-cache-result`: Will say `HIT` if the GeoWebCache WMS processed the request, and `MISS` otherwise.
- `geowebcache-miss-reason`: If the above shows as `MISS`, this will generated a short description of why the request wasn't handled by the GeoWebCache WMS.

The following are some example requests made along with the responses. These responses have been truncated to show only the information relevant for troubleshooting.

Successful request

This request was successfully handled by the GeoWebCache WMS.

Request:

```
curl -v "http://localhost:8080/geoserver/topp/wms?TILED=true&LAYERS=states&
↪FORMAT=image/png&REQUEST=GetMap&STYLES=&SRS=EPSG:4326&BBOX=-135,45,-112.5,67.5&
↪WIDTH=256&HEIGHT=256"
```

Response:

```
< HTTP/1.1 200 OK
< Content-Type: image/png
< geowebcache-crs: EPSG:4326
...
< geowebcache-layer: topp:states
< geowebcache-gridset: EPSG:4326
< geowebcache-tile-index: [2, 6, 3]
...
< geowebcache-cache-result: HIT
< geowebcache-tile-bounds: -135.0,45.0,-112.5,67.5
...
```

Wrong height parameter

The following request is not handled by the GeoWebCache WMS because the image requested (256x257) does not conform to the expected 256x256 tile size.

Request:

```
curl -v "http://localhost:8080/geoserver/topp/wms?TILED=true&LAYERS=states&
↪FORMAT=image/png&REQUEST=GetMap&STYLES=&SRS=EPSG:4326&BBOX=-135,45,-112.5,67.5&
↪WIDTH=256&HEIGHT=257"
```

Response:

```
< HTTP/1.1 200 OK
< Content-Type: image/png
< geowebcache-miss-reason: request does not align to grid(s) 'EPSG:4326'
...
```

No tile layer associated

The following request is not handled by the GeoWebCache WMS because the layer requested has no tile layer configured.

Request:

```
curl -v "http://localhost:8080/geoserver/topp/wms?TILED=true&LAYERS=tasmania_roads&
↪FORMAT=image/png&REQUEST=GetMap&STYLES=&SRS=EPSG:4326&BBOX=-135,45,-112.5,67.5&
↪WIDTH=256&HEIGHT=256"
```

Response:

```
< HTTP/1.1 200 OK
< Content-Type: image/png
< geowebcache-miss-reason: not a tile layer
...
```

Missing parameter filter

The following request is not handled by the GeoWebCache WMS because the request contains a parameter filter (BGCOLOR) that is not configured for this layer.

Request:

```
curl -v "http://localhost:8080/geoserver/topp/wms?BGCOLOR=0xAAAAAA&TILED=true&
↪LAYERS=states&FORMAT=image/png&REQUEST=GetMap&STYLES=&SRS=EPSG:4326&BBOX=-135,45,-
↪112.5,67.5&WIDTH=256&HEIGHT=256"
```

Response:

```
< HTTP/1.1 200 OK
< Content-Type: image/png
< geowebcache-miss-reason: no parameter filter exists for BGCOLOR
...
```

CRS not defined

The following request is not handled by the GeoWebCache WMS because the request references a CRS (EPSG:26986) that does not match any of the tile layer gridsets:

Request:

```
curl -v "http://localhost:8080/geoserver/topp/wms?TILED=true&LAYERS=states&
↪FORMAT=image/png&REQUEST=GetMap&STYLES=&SRS=EPSG:26986&BBOX=-135,45,-112.5,67.5&
↪WIDTH=256&HEIGHT=256"
```

Response:

```
< HTTP/1.1 200 OK
< Content-Type: image/png
< geowebcache-miss-reason: no cache exists for requested CRS
...
```

Workspace Styles

If a cached layer uses a style which is tied to a workspace, the layer needs to be viewed in the context of that workspace in order for the style to be visible. Trying to cache such a layer will result in an error.

By default, the embeded GeoWebCache uses the global workspace. This can be overridden using a `WORKSPACE` parameter. To enable this, create a List of Strings Parameter filter for the layer named `WORKSPACE`. Set the default to the name of the workspace containing the style. Setting the other values will not be useful in most cases.

Moving the style to a new workspace will require updating the filter.

This parameter only applies to integrated tile layers. If you are adding a GeoServer layer on a remote GeoServer directly to GWC, then specify the workspace as part of the path as you would normally.

CHAPTER 15

Extensions

Extensions are modules that add functionality to GeoServer. They are installed as add-ons to the base GeoServer installation.

This section describes most of the extensions available for GeoServer. Other data formats can be found in the *Vector data*, *Raster data*, and *Databases* sections.

15.1 Control flow module

The `control-flow` module for GeoServer allows the administrator to control the amount of concurrent requests actually executing inside the server, as well as giving an opportunity to slow down users making too many requests. This kind of control is useful for a number of reasons:

- *Performance*: tests show that, with local data sources, the maximum throughput in *GetMap* requests is achieved when allowing at most 2 times the number of CPU cores requests to run in parallel.
- *Resource control*: requests such as *GetMap* can use a significant amount of memory. The *WMS request limits* allow to control the amount of memory used per request, but an `OutOfMemoryError` is still possible if too many requests run in parallel. By controlling also the amount of requests executing it's possible to limit the total amount of memory used below the memory that was actually given to the Java Virtual Machine.
- *Fairness*: a single user should not be able to overwhelm the server with a lot of requests, leaving other users with tiny slices of the overall processing power.

The control flow method does not normally reject requests, it just queues up those in excess and executes them late. However, it's possible to configure the module to reject requests that have been waited in queue for too long.

15.1.1 Rule syntax reference

The current implementation of the control flow module reads its rules from a `controlflow.properties` property file located in the *GeoServer data directory*.

Total OWS request count

The global number of OWS requests executing in parallel can be specified with:

```
ows.global=<count>
```

Every request in excess will be queued and executed when other requests complete leaving some free execution slot.

Per request control

A per request type control can be demanded using the following syntax:

```
ows.<service>[.<request>[.<outputFormat>]]=<count>
```

Where:

- `<service>` is the OWS service in question (at the time of writing can be `wms`, `wfs`, `wcs`)
- `<request>`, optional, is the request type. For example, for the `wms` service it can be `GetMap`, `GetFeatureInfo`, `DescribeLayer`, `GetLegendGraphics`, `GetCapabilities`
- `<outputFormat>`, optional, is the output format of the request. For example, for the `wms GetMap` request it could be `image/png`, `image/gif` and so on

A few examples:

```
# don't allow more than 16 WCS requests in parallel
ows.wcs=16
# don't allow more than 8 GetMap requests in parallel
ows.wms.getmap=8
# don't allow more than 2 WFS GetFeature requests with Excel output format
ows.wfs.getfeature.application/msexcel=2
```

Request priority support

Requests controlled by "ows.*" controllers above can be also executed in priority order, in case there are too many the request will block and wait, and will we awoken in priority order (highest to lowest).

Currently the only way to specific a priority for a request is to add it to a request HTTP header:

```
ows.priority.http=<headerName>,<defaultPriority>
```

The header "headerName" will contain a number defining the priority for the request, the default priority is used as a fallback if/when the header is not found.

Using a header implies some other system is involved in the priority management. This is particulary good when using a load balancer, as the requests priorities need to be evenly split across cluster elements, control-flow only has visibility of a single instance. As an example, the priority will be de-facto ignored at the cluster level if there are two nodes, and for whatever chance or design, the high priority requests end up converging on the same cluster node.

Per user concurrency control

There are two mechanisms to identify user requests. The first one is cookie based, so it will work fine for browsers but not as much for other kinds of clients. The second one is ip based, which works for any type of client but that can limit all the users sitting behind the same router

This avoids a single user (as identified by a cookie) to make too many requests in parallel:

```
user=<count>
```

Where `<count>` is the maximum number of requests a single user can execute in parallel.

The following avoids a single ip address from making too many requests in parallel:

```
ip=<count>
```

Where `<count>` is the maximum number of requests a single ip address can execute in parallel.

It is also possible to make this a bit more specific and throttle a single ip address instead by using the following:

```
ip.<ip_addr>=<count>
```

Where `<count>` is the maximum number of requests the ip speficied in `<ip_addr>` will execute in parallel.

To reject requests from a list of ip addresses:

```
ip.blacklist=<ip_addr1>,<ip_addr2>,...
```

Per user rate control

The rate control rules allow to setup the maximum number of requests per unit of time, based either on a cookie or IP address. These rules look as follows (see "Per user concurrency control" for the meaning of "user" and "ip"):

```
user.ows[.<service>[.<request>[.<outputFormat>]]]=<requests>/<unit>[;<delay>s]
ip.ows[.<service>[.<request>[.<outputFormat>]]]=<requests>/<unit>[;<delay>s]
```

Where:

- `<service>` is the OWS service in question (at the time of writing can be `wms`, `wfs`, `wcs`)
- `<request>`, optional, is the request type. For example, for the `wms` service it can be `GetMap`, `GetFeatureInfo`, `DescribeLayer`, `GetLegendGraphics`, `GetCapabilities`
- `<outputFormat>`, optional, is the output format of the request. For example, for the `wms GetMap` request it could be `image/png`, `image/gif` and so on
- `<requests>` is the number of requests in the unit of time
- `<unit>` is the unit of time, can be "s", "m", "h", "d" (second, minute, hour and day respectively).
- `<delay>` is an optional the delay applied to the requests that exceed the maximum number of requests in the current time slot. If not specified, once the limit is exceeded a immediate failure response with HTTP code 429 ("Too many requests") will be sent back to the caller.

The following rule will allow 1000 WPS Execute requests a day, and delay each one in excess by 30 seconds:

```
user.ows.wps.execute=1000/d;30s
```

The following rule will instead allow up to 30 GetMap requests a second, but will immediately fail any request exceeding the cap:

```
user.ows.wms.getmap=30/s
```

In both cases headers informing the user of the request rate control will be added to the HTTP response. For example:

```
X-Rate-Limit-Context: Any OGC request
X-Rate-Limit-Limit: 10
X-Rate-Limit-Remaining: 9
X-Rate-Limit-Reset: 1103919616
X-Rate-Limit-Action: Delay excess requests 1000ms
```

In case several rate control rules apply to a single request, a batch of headers will be added to the response for each of them, it is thus advised to avoid adding too many of these rules in parallel

Where:

- `X-Rate-Limit-Context` is the type of request being subject to control
- `X-Rate-Limit-Limit` is the total amount of requests allowed in the control interval
- `X-Rate-Limit-Remaining` is the number of remaining requests allowed before the rate control kicks in
- `X-Rate-Limit-Reset` is the Unix epoch at which the new control interval will begin
- `X-Rate-Limit-Action` specifies what action is taken on requests exceeding the rate control

Timeout

A request timeout is specified with the following syntax:

```
timeout=<seconds>
```

where `<seconds>` is the number of seconds a request can stay queued waiting for execution. If the request does not enter execution before the timeout expires it will be rejected.

15.1.2 Throttling tile requests (WMS-C, TMS, WMTS)

GeoWebCache contributes three cached tiles services to GeoServer: WMS-C, TMS, and WMTS. It is also possible to use the Control flow module to throttle them, by adding the following rule to the configuration file:

```
ows.gwc=<count>
```

Where `<count>` is the maximum number of concurrent tile requests that will be delivered by GeoWebCache at any given time.

Note also that tile request are sensitive to the other rules (user based, ip based, timeout, etc).

15.1.3 A complete example

Assuming the server we want to protect has 4 cores a sample configuration could be:

```
# if a request waits in queue for more than 60 seconds it's not worth executing,
# the client will  likely have given up by then
timeout=60
# don't allow the execution of more than 100 requests total in parallel
ows.global=100
# don't allow more than 10 GetMap in parallel
ows.wms.getmap=10
# don't allow more than 4 outputs with Excel output as it's memory bound
ows.wfs.getfeature.application/msexcel=4
# don't allow a single user to perform more than 6 requests in parallel
# (6 being the Firefox default concurrency level at the time of writing)
user=6
# don't allow the execution of more than 16 tile requests in parallel
# (assuming a server with 4 cores, GWC empirical tests show that throughput
# peaks up at 4 x number of cores. Adjust as appropriate to your system)
ows.gwc=16
```

15.2 DXF OutputFormat for WFS and WPS PPIO

This extension adds two distinct functionalities to GeoServer, both related to DXF format support as an output.

DXF is a CAD interchange format, useful to import data in several CAD systems. Being a textual format it can be easily compressed to a much smaller version, so the need for a DXF-ZIP format, for low bandwidth usage.

There have been multiple revisions of the format, so we need to choose a "version" of DXF to write. The extension implements version 14, but can be easily extended (through SPI providers) to write other versions too.

The DXF OutputFormat for WFS adds the support for two additional output formats for WFS GetFeature requests. The new formats, DXF and DXF-ZIP are associated to the "application/dxf" and "application/zip" mime type, respectively. They produce a standard DXF file or a DXF file compressed in zip format.

The WPS PPIO adds dxf as an on output format option for WPS processes. The WPS PPIO requires the WPS extension to be installed on GeoServer.

15.2.1 WFS Output Format usage

Request Example:

```
http://localhost:8080/geoserver/wfs?request=GetFeature&typeName=Polygons&
outputFormat=dxf
```

Output Example (portion):

```
  0
SECTION
  2
HEADER
  9
$ACADVER
  1
AC1014
...
```

```
  0
ENDSEC
...
  0
SECTION
  2
TABLES
...
  0
TABLE
  2
LAYER
...
  0
LAYER
  5
2E
330
2
100
AcDbSymbolTableRecord
100
AcDbLayerTableRecord
  2
POLYGONS
 70
         0
 62
         7
  6
CONTINUOUS
  0
ENDTAB
...
  0
ENDSEC
  0
SECTION
  2
BLOCKS
  ...
  0
ENDSEC
  0
SECTION
  2
ENTITIES
  0
LWPOLYLINE
  5
927C0
330
1F
100
AcDbEntity
  8
POLYGONS
100
```

```
AcDbPolyline
 90
         5
 70
         1
 43
0.0
 10
500225.0
 20
500025.0
 10
500225.0
 20
500075.0
 10
500275.0
 20
500050.0
 10
500275.0
 20
500025.0
 10
500225.0
 20
500025.0
  0
ENDSEC
  0
SECTION
  2
OBJECTS
...
  0
ENDSEC
  0
EOF
```

Each single query is rendered as a layer. Geometries are encoded as entities (if simple enough to be expressed by a single DXF geometry type) or blocks (if complex, such as polygons with holes or collections).

Some options are available to control the output generated. They are described in the following paragraphs.

15.2.2 GET requests format_options

The following format_options are supported:

1. version: (number) creates a DXF in the specified version format (only 14 is currently supported)
2. asblock: (true/false) if true, all geometries are written as blocks and then inserted as entities. If false, simple geometries are directly written as entities.
3. colors: (comma delimited list of numbers): colors to be used for the DXF layers, in sequence. If layers are more than the specified colors, they will be reused many times. A set of default colors is used if the option is not used. Colors are AutoCad color numbers (7=white, etc.).
4. ltypes: (comma delimited list of line type descriptors): line types to be used for the DXF layers,

in sequence. If layers are more than the specified line types, they will be reused many times. If not specified, all layers will be given a solid, continuous line type. A descriptor has the following format: <name>!<repeatable pattern>[!<base length>], where <name> is the name assigned to the line type, <base length> (optional) is a real number that tells how long is each part of the line pattern (defaults to 0.125), and <repeatable pattern> is a visual description of the repeatable part of the line pattern, as a sequence of - (solid line),* (dot) and _ (empty space). For example a dash-dot pattern would be expressed as –_*_.

5. layers: (comma delimited list of strings) names to be assigned to the DXF layers. If specified, must contain a name for each requested query. By default a standard name will be assigned to layers.
6. withattributes: (true/false) enables writing an extra layer with attributes from each feature, the layer has a punctual geometry, with a point in the centroid of the original feature

15.2.3 POST options

Unfortunately, it's not currently possibile to use format_options in POST requests. The only thing we chose to implement is the layers options, via the handle attribute of Query attributes. So, if specified, the layer of a Query will be named as its handle attribute. The handle attribute of the GetFeature tag can also be used to override the name of the file produced.

15.2.4 WPS PPIO

When the WPS PPIO module is installed, together with the WPS extension, WPS processes returning a FeatureCollection can use application/dxf or application/zip as output mime type to get a DXF (or zipped DXF) in output.

15.3 Excel WFS Output Format

The GeoServer Excel plugin adds the ability to output WFS responses in either Excel 97-2003 (`.xls`) or Excel 2007 (`.xlsx`) formats.

15.3.1 Installation

1. Download the Excel plugin for your version of GeoServer from the download page.
2. Unzip the archive into the WEB-INF/lib directory of the GeoServer installation.
3. Restart GeoServer.

15.3.2 Usage

When making a WFS request, set the `outputFormat` to `excel` (for Excel 97-2003) or `excel2007` (for Excel 2007).

15.3.3 Examples

Excel 97-2003 GET: http://localhost:8080/geoserver/wfs?request=GetFeature&version=1.1.0&typeName=topp:states&outputFormat=excel

Excel 2007 GET: http://localhost:8080/geoserver/wfs?request=GetFeature&version=1.1.0&typeName=topp:states&outputFormat=excel2007

Excel 97-2003 POST:

```
<wfs:GetFeature service="WFS" version="1.1.0"
  outputFormat="excel"
  xmlns:topp="http://www.openplans.org/topp"
  xmlns:wfs="http://www.opengis.net/wfs"
  xmlns:ogc="http://www.opengis.net/ogc"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.opengis.net/wfs
                      http://schemas.opengis.net/wfs/1.1.0/wfs.xsd">
  <wfs:Query typeName="topp:states" />
</wfs:GetFeature>
```

15.3.4 Limitations

Excel 97-2003 files are stored in a binary format and are thus space-efficient, but have inherent size limitations (65,526 rows per sheet; 256 columns per sheet).

Excel 2007 files are XML-based, and have much higher limits (1,048,576 rows per sheet; 16,384 columns per sheet). However, because they are text files Excel 2007 files are usually larger than Excel 97-2003 files.

If the number of rows in a sheet or characters in a cell exceeds the limits of the chosen Excel file format, warning text is inserted to indicate the truncation.

15.4 GRIB

15.4.1 Adding a GRIB data store

To add a GRIB data store the user must go to *Stores –> Add New Store –> GRIB*.

Fig. 15.1: *GRIB in the list of raster data stores*

15.4.2 Configuring a GRIB data store

Option	Description
Workspace	
Data Source Name	
Description	
Enabled	
URL	

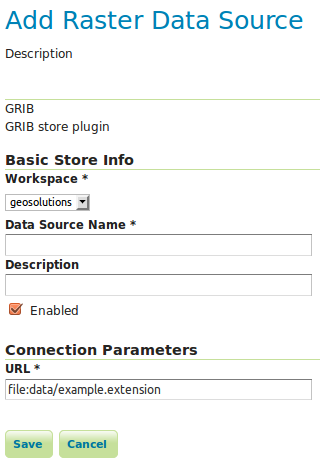


Fig. 15.2: *Configuring a GRIB data store*

Note: Note that internally the GRIB extension uses the NetCDF reader, which supports also GRIB data. See also the *NetCDF* documentation page for further information.

15.4.3 Current limitations

- Input coverages/slices should share the same bounding box (lon/lat coordinates are the same for the whole ND cube)

15.5 Imagemap

HTML ImageMaps have been used for a long time to create interactive images in a light way. Without using Flash, SVG or VML you can simply associate different links or tooltips to different regions of an image. Why can't we use this technique to achieve the same result on a GeoServer map? The idea is to combine a raster map (png, gif, jpeg, ...) with an HTML ImageMap overlay to add links, tooltips, or mouse events behavior to the map.

An example of an ImageMap adding tooltips to a map:

```
<img src="..." usemap="#mymap"/>
<map name="mymap">
     <area shape="poly" coords="536,100 535,100 534,101 533,101 532,102"  title="This␣
↪is a tooltip"/>
     <area shape="poly" coords="518,113 517,114 516,115 515,114"  title="Another␣
↪tooltip"/>
</map>
```

An example of an ImageMap adding links to a map:

```
<img src="..." usemap="#mymap"/>
<map name="mymap">
     <area shape="poly" coords="536,100 535,100 534,101 533,101 532,102"  href="http:/
↪/www.mylink.com"/>
```

```
    <area shape="poly" coords="518,113 517,114 516,115 515,114"  href="http://www.
↪mylink2.com"/>
</map>
```

A more complex example adding interactive behaviour on mouse events:

```
<img src="..." usemap="#mymap"/>
<map name="mymap">
    <area shape="poly" coords="536,100 535,100 534,101 533,101 532,102"  onmouseover=
↪"onOver('<featureid>')" onmouseout="onOut('<featureid>')"/>
    <area shape="poly" coords="518,113 517,114 516,115 515,114"   onmouseover=
↪"onOver('<featureid>')" onmouseout="onOut('<featureid>')"/>
</map>
```

To realize this in GeoServer some great community contributors developed an HTMLImageMap GetMap-Producer for GeoServer, able to render an HTMLImageMap in response to a WMS GetMap request.

The GetMapProducer is associated to the text/html mime type. It produces, for each requested layer, a <map>...</map> section containing the geometries of the layer as distinct <area> tags. Due to the limitations in the shape types supported by the <area> tag, a single geometry can be split into multiple ones. This way almost any complex geometry can be rendered transforming it into simpler ones.

To add interactive attributes we use styling. In particular, an SLD Rule containing a TextSymbolizer with a Label definition can be used to define dynamic values for the <area> tags attributes. The Rule name will be used as the attribute name.

As an example, to define a title attribute (associating a tooltip to the geometries of the layer) you can use a rule like the following one:

```
<Rule>
   <Name>title</Name>
   <TextSymbolizer>
      <Label><PropertyName>MYPROPERTY</PropertyName></Label>
   </TextSymbolizer>
</Rule>
```

To render multiple attributes, just define multiple rules, with different names (href, onmouseover, etc.)

Styling support is not limited to TextSymbolizers, you can currently use other symbolizers to detail <area> rendering. For example you can:

- use a PointSymbolizer with a Size property to define point sizes.
- use LineSymbolizer with a stroke-width CssParameter to create thick lines.

15.6 Importer

The Importer extension gives a GeoServer administrator an alternate, more-streamlined method for uploading and configuring new layers.

There are two primary advantages to using the Importer over the standard GeoServer data-loading workflow:

1. **Supports batch operations** (loading and publishing multiple spatial files or database tables in one operation)
2. **Creates unique styles** for each layer, rather than linking to the same (existing) styles.

This section will discuss the Importer extension.

15.6.1 Installing the Importer extension

The Importer extension is an official extension, available on the GeoServer download page.

1. Download the extension for your version of GeoServer. (If you see an option, select *Core*.)

   Warning: Make sure to match the version of the extension to the version of GeoServer.

2. Extract the archive and copy the contents into the GeoServer `WEB-INF/lib` directory.
3. Restart GeoServer.
4. To verify that the extension was installed successfully, open the *Web administration interface* and look for an *Import Data* option in the *Data* section on the left-side menu.

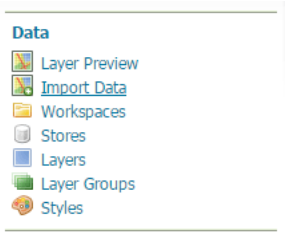


Fig. 15.3: Importer extension successfully installed.

For additional information please see the section on *Using the Importer extension*.

15.6.2 Using the Importer extension

Here are step-by-step instructions to import multiple shapefiles in one operation. For more details on different types of operations, please see the *Importer interface reference*

1. Find a directory of shapefiles and copy into your *GeoServer data directory*.

   Note: You can always use the Natural Earth Quickstart data for this task.

2. Log in as an administrator and navigate to the *Data –> Import Data* page.
3. For select *Spatial Files* as the data source.

Fig. 15.4: Data source

4. Click *Browse* to navigate to the directory of shapefiles to be imported.
5. The web-based file browser will show as options your home directory, data directory, and the root of your file system (or drive). In this case, select *Data directory*

Fig. 15.5: Directory

6. Back on the main form, select *Create new* next to *Workspace*, and enter `ne` to denote the workspace.

Note: Make sure the *Store* field reads *Create new* as well.

Fig. 15.6: Import target workspace

7. Click *Next* to start the import process.
8. On the next screen, any layers available for import will be shown.

Note: Non-spatial files will be ignored.

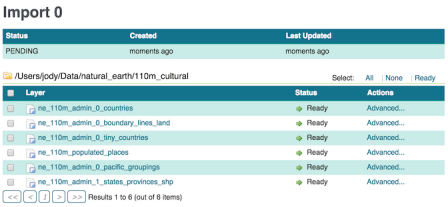


Fig. 15.7: Import layer list

9. In most cases, all files will be ready for import, but if the the spatial reference system (SRS) is not recognized, you will need to manually input this but clicking *Advanced*

Note: You will need to manually input the SRS if you used the Natural Earth data above. For each layer click on *Advanced* and set reprojection to `EPSG:4326`.

10. Check the box next to each layer you wish to import.
11. When ready, click *Import*.

Warning: Don't click *Done* at this point, otherwise the import will be canceled.

12. The results of the import process will be shown next to each layer.
13. When finished, click *Done*.

Note: Recent import processes are listed at the bottom of the page. You may wish to visit these pages to check if any difficulties were encountered during the import process or import additional layers.

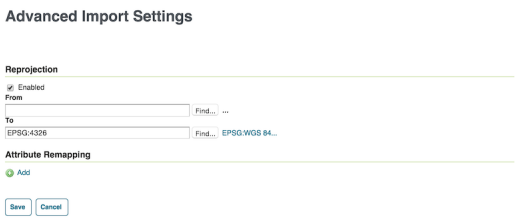


Fig. 15.8: Advanced import settings

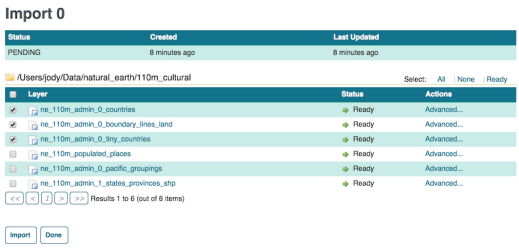


Fig. 15.9: Setting the layers to import

15.6.3 Importer interface reference

The Layer Importer user interface is a component of the GeoServer web interface. You can access it from the GeoServer web interface by clicking the *Import Data* link, found on the left side of the screen after logging in.

Data sources page

The front page of the Layer Importer is where the data source and format are set. The following options are displayed:

Choose a data source to import from

Select one of the following data sources to use for the import:

- *Spatial Files* (see *Supported data formats* for more details)
- *PostGIS* database
- *Oracle* database
- *SQL Server* database

The contents of the next section is dependent on the data source chosen here.

Fig. 15.10: Recent imports

Fig. 15.11: Choose a data source

Configure the data source: Spatial Files

There is a single box for selecting a file or directory. Click the *Browse* link to bring up a file chooser. To select a file, click on it. To select a directory, click on a directory name to open it and then click *OK*.

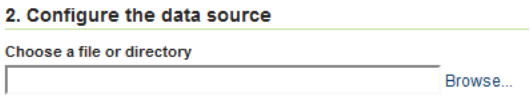


Fig. 15.12: Spatial file data source

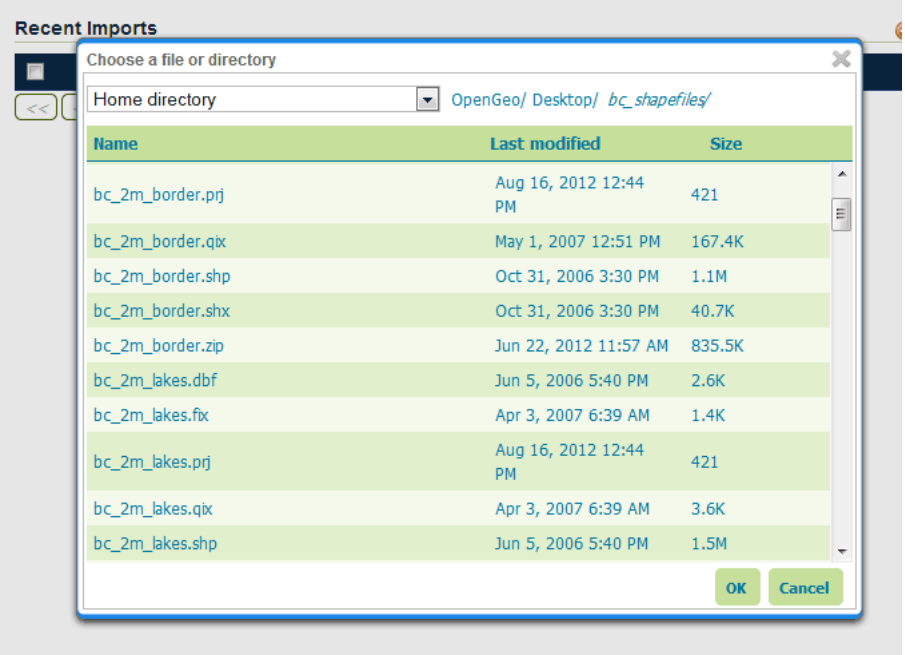


Fig. 15.13: File chooser for selecting spatial files

Configure the data source: PostGIS

Fill out fields for *Connection type* (Default or JNDI) *Host*, *Port*, *Database* name, *Schema*, *Username* to connect with, and *Password*.

There are also advanced connection options, which are common to the standard PostGIS store loading procedure. (See the PostGIS data store page in the GeoServer reference documentation.)

Fig. 15.14: PostGIS data source connection

Configure the data source: Oracle

The parameter fields for the Oracle import are identical to that of PostGIS. The fields aren't populated with default credentials with the exception of the port, which is set to **1521** by default.

Note: This option is only enabled if the *Oracle* extension is installed.

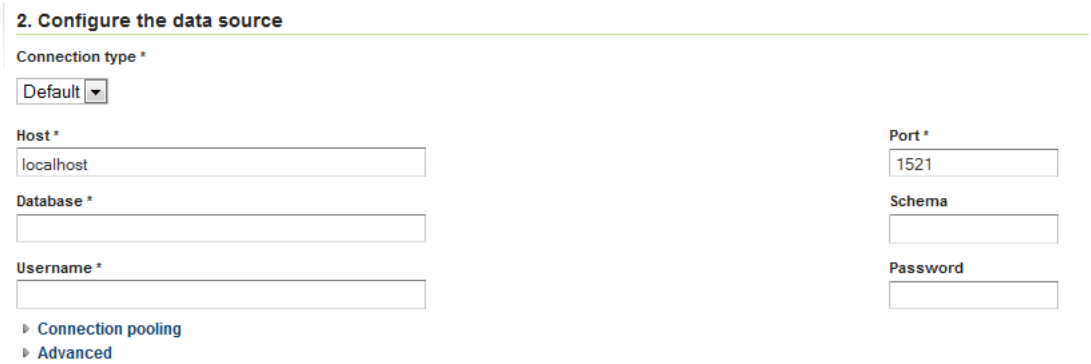


Fig. 15.15: Oracle data source connection

Configure the data source: SQL Server

The parameter fields for the SQL Server import are identical to that of PostGIS. The fields aren't populated with default credentials with the exception of the port, which is set to **4866** by default.

Note: This option is only enabled if the *SQL Server* extension is installed.

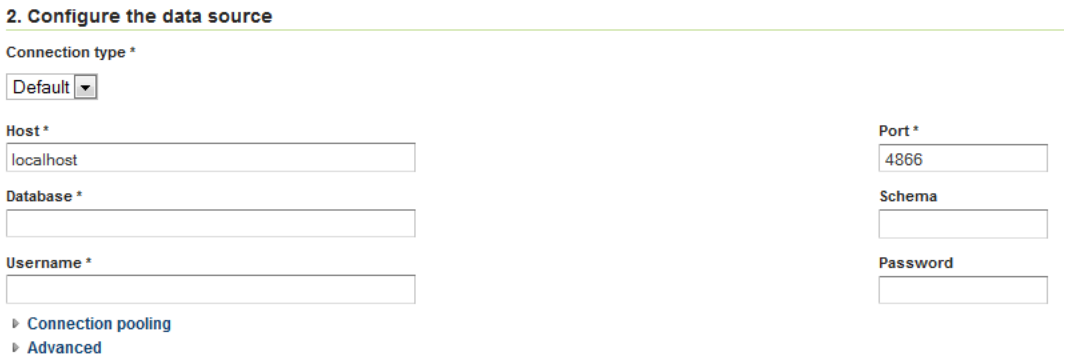


Fig. 15.16: SQL Server data source connection

Specify the target for the import

This area specifies where in the GeoServer catalog the new data source will be stored. This does not affect file placement.

Select the name of an existing workspace and store.

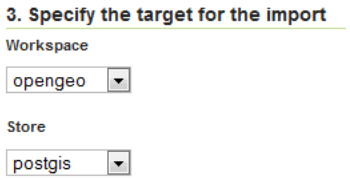


Fig. 15.17: Target workspace and store in GeoServer

Alternately, select *Create New* and type in a names for a new workspace or store. During the import process, these will be created.

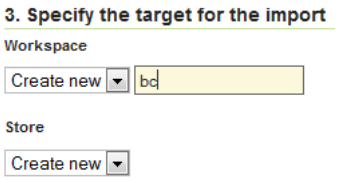


Fig. 15.18: Creating a new workspace and store

Recent imports

This section will list previous imports, and whether they were successful or not. Items can be removed from this list with the *Remove* button, but otherwise cannot be edited.

When ready to continue to the next page, click *Next*.

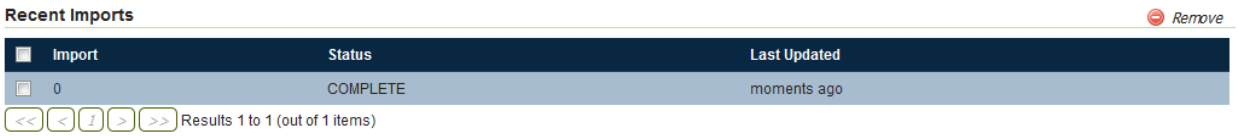


Fig. 15.19: Recent imports

Layer listing page

On the next page will be a list of layers found by the Layer Importer. The layers will be named according to the source content's name (file name of database table name). For each entry there will be a *Status* showing if the source is ready to be imported.

All layers will be selected for import by default, but can be deselected here by unchecking the box next to each entry.

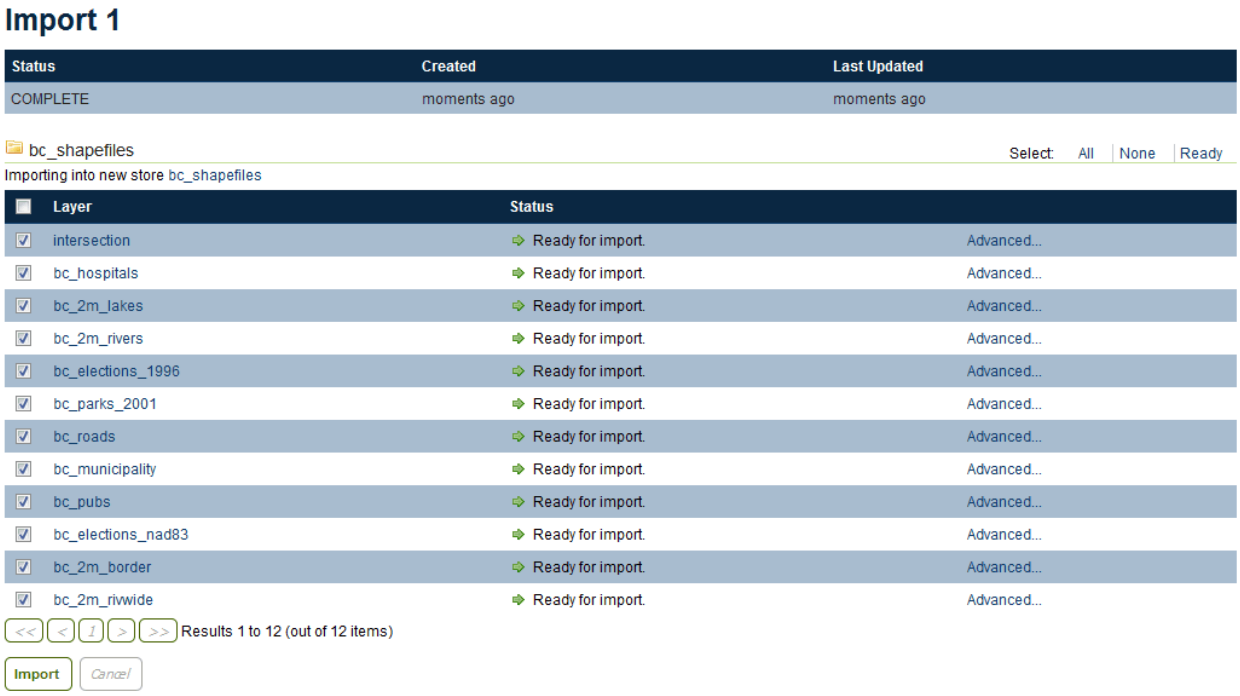


Fig. 15.20: List of layers to be imported

A common issue during the import process is when a CRS cannot be determined for a given layer. In this case, a dialog box will display where the CRS can be declared explicitly. Enter the CRS and Click *Apply*.

Fig. 15.21: Declaring a CRS

When ready to perform the import, click *Import*.

Each selected layer will be added to the GeoServer catalog inside a new or existing store, and published as a layer.

After the import is complete the status area will refresh showing if the import was successful for each layer. If successful, a dialog box for previewing the layer will be displayed, with options for *Layer Preview* (OpenLayers), *Google Earth*, and *GeoExplorer*.

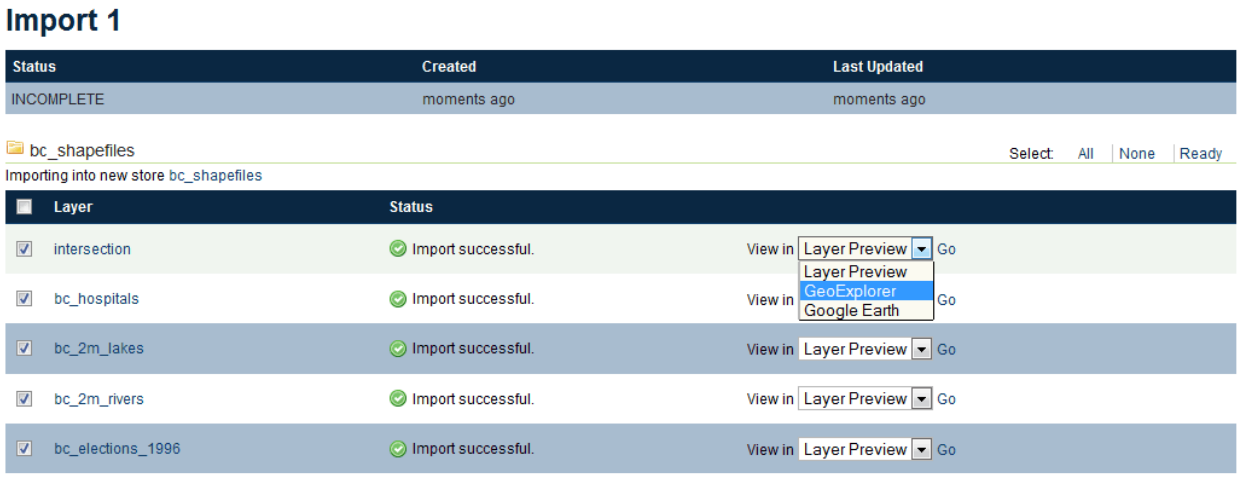


Fig. 15.22: Layers successfully imported

Advanced import settings page

The *Advanced* link next to each layer will lead to the Advanced import settings page.

On this page, data can be set to be reprojected from one CRS to another during the import process. To enable reprojection, select the *Reprojection* box, and enter the source and target CRS.

In addition, on this page attributes can be renamed and their type changed. Click on the *Add* link under *Attribute Remapping* to select the attribute to alter, its type, and its new name. Click *Apply* when done.

Click *Save* when finished.

15.6.4 Supported data formats

The importer supports any format that GeoServer can use a data store or coverage store. These include the most commonly used formats:

- Shapefile
- GeoTIFF

And a few additional formats:

- CSV
- KML

The following databases are supported:

- PostGIS
- Oracle

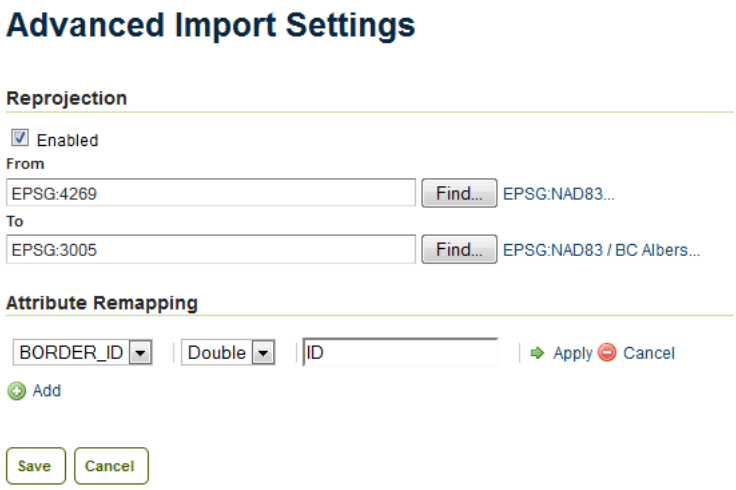


Fig. 15.23: Advanced layer list page

- Microsoft SQL Server

Note: Oracle and SQL Server require extra drivers to be installed.

- *Install instructions for Oracle*
- *Install instructions for SQL Server*

15.6.5 REST API

Importer concepts

The importer REST api is built around a tree of objects representing a single import, structured as follows:

- **import**
  - target workspace
  - data
  - **task (one or more)**
    - data
    - layer
    - transformation (one or more)

An **import** refers to the top level object and is a "session" like entity the state of the entire import. It maintains information relevant to the import as a whole such as user infromation, timestamps along with optional information that is uniform along all tasks, such as a target workspace, the shared input data (e.g., a directory, a database). An import is made of any number of task objects.

A **data** is the description of the source data of a import (overall) or a task. In case the import has a global data definition, this normally refers to an aggregate store such as a directory or a database, and the data associated to the tasks refers to a single element inside such aggregation, such as a single file or table.

A **task** represents a unit of work to the importer needed to register one new layer, or alter an existing one, and contains the following information:

- The data being imported
- The target store that is the destination of the import
- The target layer
- The data of a task, referred to as its source, is the data to be processed as part of the task.
- The transformations that we need to apply to the data before it gets imported

This data comes in a variety of forms including:

- A spatial file (Shapefile, GeoTiff, KML, etc. . .)
- A directory of spatial files
- A table in a spatial database
- A remote location that the server will download data from

A task is classified as either "direct" or "indirect". A *direct task* is one in which the data being imported requires no transformation to be imported. It is imported directly. An example of such a task is one that involves simply importing an existing Shapefile as is. An *indirect task* is one that does require a **transformation** to the original import data. An example of an indirect task is one that involves importing a Shapefile into an existing PostGIS database. Another example of indirect task might involve taking a CSV file as an input, turning a x and y column into a Point, remapping a string column into a timestamp, and finally import the result into a PostGIS.

REST API Reference

All the imports

/imports

Method	Action	Status Code/Headers	Input	Output	Parameters
GET	Retrieve all imports	200	n/a	Import Collection	n/a
POST	Create a new import	201 with Location header	n/a	Imports	async=false/true,execute=false/true

Retrieving the list of all imports

```
GET /imports
```

results in:

```
Status: 200 OK
Content-Type: application/json

        {
           "imports": [{
             "id": 0,
```

```
            "state": "COMPLETE",
            "href": "http://localhost:8080/geoserver/rest/imports/0"

          }, {
            "id": 1,
            "state": "PENDING",
            "href": "http://localhost:8080/geoserver/rest/imports/1"
          }]
        }
```

Creating a new import

Posting to the /imports path a import json object creates a new import session:

```
Content-Type: application/json

{
   "import": {
      "targetWorkspace": {
         "workspace": {
            "name": "scratch"
         }
      },
      "targetStore": {
         "dataStore": {
            "name": "shapes"
         }
      },
      "data": {
        "type": "file",
        "file": "/data/spearfish/archsites.shp"
      }
   }
}
```

The parameters are:

Name	Optional	Description
targetWorkspace	Y	The target workspace to import to
targetStore	Y	The target store to import to
data	Y	The data to be imported

The mere creation does not start the import, but it may automatically populate its tasks depending on the target. For example, by referring a directory of shapefiles to be importer, the creation will automatically fill in a task to import each of the shapefiles as a new layer.

The response to the above POST request will be:

```
Status: 201 Created
Location: http://localhost:8080/geoserver/rest/imports/2
Content-Type: application/json

{
  "import": {
    "id": 2,
```

```
    "href": "http://localhost:8080/geoserver/rest/imports/2",
    "state": "READY",
    "targetWorkspace": {
      "workspace": {
        "name": "scratch"
      }
    },
    "targetStore": {
      "dataStore": {
        "name": "shapes",
        "type": "PostGIS"
      }
    },
    "data": {
      "type": "file",
      "format": "Shapefile",
      "href": "http://localhost:8080/geoserver/rest/imports/2/data",
      "file": "archsites.shp"
    },
    "tasks": [
      {
        "id": 0,
        "href": "http://localhost:8080/geoserver/rest/imports/2/tasks/0",
        "state": "READY"
      }
    ]
  }
}
```

The operation of populating the tasks can require time, especially if done against a large set of files, or against a "remote" data (more on this later), in this case the POST request can include `?async=true` at the end of the URL to make the importer run it asynchronously. In this case the import will be created in INIT state and will remain in such state until all the data transfer and task creation operations are completed. In case of failure to fetch data the import will immediately stop, the state will switch to the `INIT_ERROR` state, and a error message will appear in the import context "message" field.

Adding the "execute=true" parameter to the context creation will also make the import start immediately, assuming tasks can be created during the init phase. Combining both execute and async, "?async=true&execute=true" will make the importer start an asynchronous initialization and execution.

The import can also have a list of default transformations, that will be applied to tasks as they get created, either out of the initial data, or by upload. Here is an example of a import context creation with a default transformation:

```
{
  "import": {
    "targetWorkspace": {
      "workspace": {
        "name": "topp"
      }
    },
    "data": {
      "type": "file",
      "file": "/tmp/locations.csv"
    },
    "targetStore": {
      "dataStore": {
        "name": "h2"
```

```
      }
    },
    "transforms": [
      {
        "type": "AttributesToPointGeometryTransform",
        "latField": "LAT",
        "lngField": "LON"
      }
    ]
  }
}
```

To get more information about transformations see the *Transformation reference*.

Import object

/imports/<importId>

Method	Action	Status Code/Headers	Input	Output	Parameters
GET	Retrieve import with id <importId>	200	n/a	Imports	n/a
POST	Execute import with id <importId>	204	n/a	n/a	async=true/false
PUT	Create import with proposed id <importId>. If the proposed id is ahead of the current (next) id, the current id will be advanced. If the proposed id is less than or equal to the current id, the current will be used. This allows an external system to dictate the id management.	201 with Location header	n/a	Imports	n/a
DELETE	Remove import with id <importId>	200	n/a	n/a	n/a

The representation of a import is the same as the one contained in the import creation response. The execution of a import can be a long task, as such, it's possible to add `async=true` to the request to make it run in a asynchronous fashion, the client will have to poll the import representation and check when it reaches the "COMPLETE" state.

Data

A import can have a "data" representing the source of the data to be imported. The data can be of different types, in particular, "file", "directory", "mosaic", "database" and "remote". During the import initialization the importer will scan the contents of said resource, and generate import tasks for each data found in it.

Most data types are discussed in the task section, the only type that's specific to the whole import context is the "remote" one, that is used to ask the importer to fetch the data from a remote location autonomously, without asking the client to perform an upload.

The representation of a remote resource looks as follows:

```
"data": {
  "type": "remote",
  "location": "ftp://fthost/path/to/importFile.zip",
```

```
  "username": "user",
  "password": "secret",
  "domain" : "mydomain"
}
```

The location can be any URI supported by Commons VFS, including HTTP and FTP servers. The `username`, `password` and `domain` elements are all optional, and required only if the remote server demands an authentication of sorts. In case the referred file is compressed, it will be unpacked as the download completes, and the tasks will be created over the result of unpacking.

Tasks

/imports/<importId>/tasks

Method	Action	Status Code/Headers	Input	Output
GET	Retrieve all tasks for import with id <importId>	200	n/a	Task Collection
POST	Create a new task	201 with Location header	*Multipart form data*	Tasks

Getting the list of tasks

```
GET /imports/0/tasks
```

Results in:

```
Status: 200 OK
Content-Type: application/json

{
  "tasks": [
    {
      "id": 0,
      "href": "http://localhost:8080/geoserver/rest/imports/2/tasks/0",
      "state": "READY"
    }
  ]
}
```

Creating a new task as a file upload

A new task can be created by issuing a POST to `imports/<importId>/tasks` as a "Content-type: multipart/form-data" multipart encoded data as defined by RFC 2388. One or more file can be uploaded this way, and a task will be created for importing them. In case the file being uploaded is a zip file, it will be unzipped on the server side and treated as a directory of files.

The response to the upload will be the creation of a new task, for example:

```
Status: 201 Created
Location: http://localhost:8080/geoserver/rest/imports/1/tasks/1
Content-type: application/json

{
  "task": {
    "id": 1,
    "href": "http://localhost:8080/geoserver/rest/imports/2/tasks/1",
    "state": "READY",
    "updateMode": "CREATE",
    "data": {
      "type": "file",
      "format": "Shapefile",
      "href": "http://localhost:8080/geoserver/rest/imports/2/tasks/1/data",
      "file": "bugsites.shp"
    },
    "target": {
      "href": "http://localhost:8080/geoserver/rest/imports/2/tasks/1/target",
      "dataStore": {
        "name": "shapes",
        "type": "PostGIS"
      }
    },
    "progress": "http://localhost:8080/geoserver/rest/imports/2/tasks/1/progress",
    "layer": {
      "name": "bugsites",
      "href": "http://localhost:8080/geoserver/rest/imports/2/tasks/1/layer"
    },
    "transformChain": {
      "type": "vector",
      "transforms": []
    }
  }
}
```

Creating a new task from form upload

This creation mode assumes the POST to `imports/<importId>/tasks` of form url encoded data containing a `url` parameter:

```
Content-type: application/x-www-form-urlencoded

url=file:///data/spearfish/
```

The creation response will be the same as the multipart upload.

Single task resource

/imports/<importId>/task/<taskId>

Method	Action	Status Code/Headers	Input	Output
GET	Retrieve task with id <taskId> within import with id <importId>	200	n/a	Task
PUT	Modify task with id <taskId> within import with id <importId>	200	Task	Task
DELETE	Remove task with id <taskId> within import with id <importId>	200	n/a	n/a

The representation of a task resource is the same one reported in the task creation response.

Updating a task

A PUT request over an existing task can be used to update its representation. The representation can be partial, and just contains the elements that need to be updated.

The updateMode of a task normally starts as "CREATE", that is, create the target resource if missing. Other possible values are "REPLACE", that is, delete the existing features in the target layer and replace them with the task source ones, or "APPEND", to just add the features from the task source into an existing layer.

The following PUT request updates a task from "CREATE" to "APPEND" mode:

```
Content-Type: application/json

{
  "task": {
     "updateMode": "APPEND"
  }
}
```

Directory files representation

The following operations are specific to data objects of type `directory`.

/imports/<importId>/task/<taskId>/data/files

Method	Action	Status Code/Headers	Input	Output
GET	Retrieve the list of files for a task with id <taskId> within import with id <importId>	200	n/a	Task

The response to a GET request will be:

```
Status: 200 OK
Content-Type: application/json
```

```
{
        files: [
                {
                file: "tasmania_cities.shp",
                href: "http://localhost:8080/geoserver/rest/imports/0/tasks/0/data/
↪files/tasmania_cities.shp"
                },
                {
                file: "tasmania_roads.shp",
                href: "http://localhost:8080/geoserver/rest/imports/0/tasks/0/data/
↪files/tasmania_roads.shp"
                },
                {
                file: "tasmania_state_boundaries.shp",
                href: "http://localhost:8080/geoserver/rest/imports/0/tasks/0/data/
↪files/tasmania_state_boundaries.shp"
                },
                {
                file: "tasmania_water_bodies.shp",
                href: "http://localhost:8080/geoserver/rest/imports/0/tasks/0/data/
↪files/tasmania_water_bodies.shp"
                }
        ]
}
```

/imports/<importId>/task/<taskId>/data/files/<fileId>

Method	Action	Status Code/Headers	Input	Output
GET	Retrieve the file with id <fileId> from the data of a task with id <taskId> within import with id <importId>	200	n/a	Task
DELETE	Remove a specific file from the task with id <taskId> within import with id <importId>	200	n/a	n/a

Following the links we'll get to the representation of a single file, notice how in this case a main file can be associate to sidecar files:

```
Status: 200 OK
Content-Type: application/json

{
        type: "file",
        format: "Shapefile",
        location: "C:\devel\gs_data\release\data\taz_shapes",
        file: "tasmania_cities.shp",
        href: "http://localhost:8080/geoserver/rest/imports/0/tasks/0/data/files/
↪tasmania_cities.shp",
        prj: "tasmania_cities.prj",
        other: [
                "tasmania_cities.dbf",
                "tasmania_cities.shx"
        ]
}
```

Mosaic extensions

In case the input data is of `mosaic` type, we have all the attributes typical of a directory, plus support for directly specifying the timestamp of a particular granule.

In order to specify the timestamp a PUT request can be issued against the granule:

```
Content-Type: application/json

{
   "timestamp": "2004-01-01T00:00:00.000+0000"
}
```

and the response will be:

```
Status: 200 OK
Content-Type: application/json

{
  "type": "file",
  "format": "GeoTIFF",
  "href": "http://localhost:8080/geoserver/rest/imports/0/tasks/0/data/files/bm_
↪200401.tif",
  "location": "/data/bluemarble/mosaic",
  "file": "bm_200401.tiff",
  "prj": null,
  "other": [],
  "timestamp": "2004-01-01T00:00:00.000+0000"
}
```

Database data

The following operations are specific to data objects of type `database`. At the time or writing, the REST API does not allow the creation of a database data source, but it can provide a read only description of one that has been created using the GUI.

/imports/<importId>/tasks/<taskId>/data

Method	Action	Status Code/Headers	Input	Output
GET	Retrieve the database connection parameters for a task with id <taskId> within import with id <importId>	200	n/a	List of database connection parameters and available tables

Performing a GET on a database type data will result in the following response:

```
{
        type: "database",
        format: "PostGIS",
        href: "http://localhost:8080/geoserver/rest/imports/0/data",
        parameters: {
                schema: "public",
                fetch size: 1000,
```

```
                validate connections: true,
                Connection timeout: 20,
                Primary key metadata table: null,
                preparedStatements: true,
                database: "gttest",
                port: 5432,
                passwd: "cite",
                min connections: 1,
                dbtype: "postgis",
                host: "localhost",
                Loose bbox: true,
                max connections: 10,
                user: "cite"
        },
        tables: [
                "geoline",
                "geopoint",
                "lakes",
                "line3d",
        ]
}
```

Database table

The following operations are specific to data objects of type `table`. At the time or writing, the REST API does not allow the creation of a database data source, but it can provide a read only description of one that has been created using the GUI. A table description is normally linked to task, and refers to a database data linked to the overall import.

/imports/<importId>/tasks/<taskId>/data

Method	Action	Status Code/Headers	Input	Output
GET	Retrieve the table description for a task with id <taskId> within import with id <importId>	200	n/a	A table representation

Performing a GET on a database type data will result in the following response:

```
{
        type: "table",
        name: "abc",
        format: "PostGIS",
        href: "http://localhost:8080/geoserver/rest/imports/0/tasks/0/data"
}
```

Task target layer

/imports/<importId>/tasks/<taskId>/layer

The layer defines how the target layer will be created

Method	Action	Status Code/Headers	Input	Output
GET	Retrieve the layer of a task with id <taskId> within import with id <importId>	200	n/a	A layer JSON representation
PUT	Modify the target layer for a task with id <taskId> within import with id <importId>	200	Task	Task

Requesting the task layer will result in the following:

```
Status: 200 OK
Content-Type: application/json

{
        layer: {
        name: "tasmania_cities",
        href: "http://localhost:8080/geoserver/rest/imports/0/tasks/0/layer",
        title: "tasmania_cities",
        originalName: "tasmania_cities",
        nativeName: "tasmania_cities",
        srs: "EPSG:4326",
        bbox: {
                minx: 147.2909004483,
                miny: -42.85110181689001,
                maxx: 147.2911004483,
                maxy: -42.85090181689,
                crs: "GEOGCS["WGS 84", DATUM["World Geodetic System 1984", SPHEROID[
↪"WGS 84", 6378137.0, 298.257223563, AUTHORITY["EPSG","7030"]], AUTHORITY["EPSG",
↪"6326"]], PRIMEM["Greenwich", 0.0, AUTHORITY["EPSG","8901"]], UNIT["degree", 0.
↪0174532925199433295], AXIS["Geodetic longitude", EAST], AXIS["Geodetic latitude",␣
↪NORTH], AUTHORITY["EPSG","4326"]]"
        },
        attributes: [
                {
                        name: "the_geom",
                        binding: "org.locationtech.jts.geom.MultiPoint"
                },
                {
                        name: "CITY_NAME",
                        binding: "java.lang.String"
                },
                {
                        name: "ADMIN_NAME",
                        binding: "java.lang.String"
                },
                {
                        name: "CNTRY_NAME",
                        binding: "java.lang.String"
                },
                {
                        name: "STATUS",
                        binding: "java.lang.String"
                },
                {
                        name: "POP_CLASS",
                        binding: "java.lang.String"
                }
                ],
```

```
                style: {
                        name: "cite_tasmania_cities",
                        href: "http://localhost:8080/geoserver/rest/imports/0/tasks/0/
↪layer/style"
                }
        }
}
```

All the above attributes can be updated using a PUT request. Even if the above representation is similar to the REST config API, it should not be confused with it, as it does not support all the same properties, in particular the supported properties are all the ones listed above.

Task transformations

/imports/<importId>/tasks/<taskId>/transforms

Method	Action	Status Code/Headers	Input	Output
GET	Retrieve the list of transformations of a task with id <taskId> within import with id <importId>	200	n/a	A list of transfromations in JSON format
POST	Create a new transormation and append it inside a task with id <taskId> within import with id <importId>	201	A JSON transformation representation	The transform location

Retrieving the transformation list

A GET request for the list of transformations will result in the following response:

```
Status: 200 OK
Content-Type: application/json

{
  "transforms": [
    {
      "type": "ReprojectTransform",
      "href": "http://localhost:8080/geoserver/rest/imports/0/tasks/1/transforms/0",
      "source": null,
      "target": "EPSG:4326"
    },
    {
      "type": "DateFormatTransform",
      "href": "http://localhost:8080/geoserver/rest/imports/0/tasks/1/transforms/1",
      "field": "date",
      "format": "yyyyMMdd"
    }
  ]
}
```

Appending a new transformation

Creating a new transformation requires posting a JSON document with a `type` property identifying the class of the transformation, plus any extra attribute required by the transformation itself (this is transformation specific, each one will use a different set of attributes).

The following POST request creates an attribute type remapping:

```
Content-Type: application/json

{
   "type": "AttributeRemapTransform",
   "field": "cat",
   "target": "java.lang.Integer"
}
```

The response will be:

```
Status: 201 OK
Location: http://localhost:8080/geoserver/rest/imports/0/tasks/1/transform/2
```

/imports/<importId>/tasks/<taskId>/transforms/<transformId>

Method	Action	Status Code/Headers	Input	Output
GET	Retrieve a transformation identified by <transformId> inside a task with id <taskId> within import with id <importId>	200	n/a	A single transformation in JSON format
PUT	Modifies the definition of a transformation identified by <transformId> inside a task with id <taskId> within import with id <importId>	200	A JSON transformation representation (eventually just the portion of it that needs to be modified)	The full transformation representation
DELETE	Removes the transformation identified by <transformId> inside a task with id <taskId> within import with id <importId>	200	A JSON transformation representation (eventually just the portion of it that needs to be modified)	The full transformation representation

Retrieve a single transformation

Requesting a single transformation by identifier will result in the following response:

```
Status: 200 OK
Content-Type: application/json

{
  "type": "ReprojectTransform",
  "href": "http://localhost:8080/geoserver/rest/imports/0/tasks/1/transforms/0",
  "source": null,
```

```
  "target": "EPSG:4326"
}
```

Modify an existing transformation

Assuming we have a reprojection transformation, and that we need to change the target SRS type, the following PUT request will do the job:

```
Content-Type: application/json
{
   "type": "ReprojectTransform",
   "target": "EPSG:3005"
}
```

The response will be:

```
Status: 200 OK
    Content-Type: application/json

    {
      "type": "ReprojectTransform",
      "href": "http://localhost:8080/geoserver/rest/imports/0/tasks/1/transform/0",
      "source": null,
      "target": "EPSG:3005"
    }
```

Transformation reference

AttributeRemapTransform

Remaps a certain field to a given target data type

Parameter	Optional	Description
field	N	The name of the field to be remapped
target	N	The "target" field type, as a fully qualified Java class name

AttributeComputeTransform

Computes a new field based on an expression that can use the other field values

Parameter	Optional	Description
field	N	The name of the field to be computed
fieldType	N	The field type, as a fully qualified Java class name (e.g., `java.lang.String`, `java.lang.Integer`, `java.util.Date` and so on)
cql	N	The (E)CQL expression used to compute the new field (can be a constant value, e.g. `'My String'`)

AttributesToPointGeometryTransform

Transforms two numeric fields `latField` and `lngField` into a point geometry representation `POINT(lngField,latField)`, the source fields will be removed.

Parameter	Optional	Description
latField	N	The "latitude" field
lngField	N	The "longitude" field

CreateIndexTransform

For database targets only, creates an index on a given column after importing the data into the database

Parameter	Optional	Description
field	N	The field to be indexed

DateFormatTransform

Parses a string representation of a date into a Date/Timestamp object

Parameter	Optional	Description
field	N	The field to be parsed
format	Y	A date parsing pattern, setup using the Java SimpleDateFormat syntax. In case it's missing, a number of built-in formats will be tried instead (short and full ISO date formats, dates without any separators).
enddate	Y	The field used as end date for the time dimension.
presentation	Y	The time dimension presentation type; one of {LIST; DISCRETE_INTERVAL; CONTINUOUS_INTERVAL}

IntegerFieldToDateTransform

Takes a integer field and transforms it to a date, interpreting the intereger field as a date

Parameter	Optional	Description
field	N	The field containing the year information

ReprojectTransform

Reprojects a vector layer from a source CRS to a target CRS

Parameter	Optional	Description
source	Y	Identifier of the source coordinate reference system (the native one will be used if missing)
target	N	Identifier of the target coordinate reference system

GdalTranslateTransform

Applies `gdal_translate` to a single file raster input. Requires `gdal_translate` to be inside the PATH used by the web container running GeoServer.

Parameter	Optional	Description
options	N	Array of options that will be passed to `gdal_translate` (beside the input and output names, which are internally managed)

GdalWarpTransform

Applies `gdalwarp` to a single file raster input. Requires `gdalwarp` to be inside the PATH used by the web container running GeoServer.

Parameter	Optional	Description
options	N	Array of options that will be passed to `gdalwarp` (beside the input and output names, which are internally managed)

GdalAddoTransform

Applies `gdaladdo` to a single file raster input. Requires `gdaladdo` to be inside the PATH used by the web container running GeoServer.

Parameter	Optional	Description
options	N	Array of options that will be passed to `gdaladdo` (beside the input file name, which is internally managed)
levels	N	Array of integers with the overview levels that will be passed to `gdaladdo`

PostScriptTransform

Runs the specified script after the data is imported. The script must be located in `$GEOSERVER_DATA_DIR/importer/scripts`. The script can be any executable file. At the time of writing, there is no way to pass information about the data just imported to the script (TBD).

Parameter	Optional	Description
name	N	Name of the script to be invoked
options	Y	Array of options that will be passed to the script

15.6.6 Importer REST API examples

Mass configuring a directory of shapefiles

In order to initiate an import of the `c:\data\tasmania` directory into the existing `tasmania` workspace the following JSON will be POSTed to GeoServer:

```
{
   "import": {
      "targetWorkspace": {
         "workspace": {
            "name": "tasmania"
         }
      },
      "data": {
        "type": "directory",
        "location": "C:/data/tasmania"
      }
   }
}
```

This curl command can be used for the purpose:

```
curl -u admin:geoserver -XPOST -H "Content-type: application/json" -d @import.json
↪"http://localhost:8080/geoserver/rest/imports"
```

The importer will locate the files to be imported, and automatically prepare the tasks, returning the following response:

```
{
  "import": {
    "id": 9,
    "href": "http://localhost:8080/geoserver/rest/imports/9",
    "state": "PENDING",
    "archive": false,
    "targetWorkspace": {
      "workspace": {
        "name": "tasmania"
      }
    },
    "data": {
      "type": "directory",
      "format": "Shapefile",
      "location": "C:\\data\\tasmania",
      "href": "http://localhost:8080/geoserver/rest/imports/9/data"
    },
    "tasks": [
      {
        "id": 0,
        "href": "http://localhost:8080/geoserver/rest/imports/9/tasks/0",
        "state": "READY"
      },
      {
        "id": 1,
        "href": "http://localhost:8080/geoserver/rest/imports/9/tasks/1",
        "state": "READY"
      },
      {
```

```
        "id": 2,
        "href": "http://localhost:8080/geoserver/rest/imports/9/tasks/2",
        "state": "READY"
      },
      {
        "id": 3,
        "href": "http://localhost:8080/geoserver/rest/imports/9/tasks/3",
        "state": "READY"
      }
    ]
  }
}
```

After checking every task is ready, the import can be initiated by executing a POST on the import resource:

```
curl -u admin:geoserver -XPOST "http://localhost:8080/geoserver/rest/imports/9"
```

The resource can then be monitored for progress, and eventually final results:

```
curl -u admin:geoserver -XGET "http://localhost:8080/geoserver/rest/imports/9"
```

Which in case of successful import will look like:

```
{
  "import": {
    "id": 9,
    "href": "http://localhost:8080/geoserver/rest/imports/9",
    "state": "COMPLETE",
    "archive": false,
    "targetWorkspace": {
      "workspace": {
        "name": "tasmania"
      }
    },
    "data": {
      "type": "directory",
      "format": "Shapefile",
      "location": "C:\\data\\tasmania",
      "href": "http://localhost:8080/geoserver/rest/imports/9/data"
    },
    "tasks": [
      {
        "id": 0,
        "href": "http://localhost:8080/geoserver/rest/imports/9/tasks/0",
        "state": "COMPLETE"
      },
      {
        "id": 1,
        "href": "http://localhost:8080/geoserver/rest/imports/9/tasks/1",
        "state": "COMPLETE"
      },
      {
        "id": 2,
        "href": "http://localhost:8080/geoserver/rest/imports/9/tasks/2",
        "state": "COMPLETE"
      },
      {
        "id": 3,
```

```
        "href": "http://localhost:8080/geoserver/rest/imports/9/tasks/3",
        "state": "COMPLETE"
      }
    ]
  }
}
```

Configuring a shapefile with no projection information

In this case, let's assume we have a single shapefile, tasmania_cities.shp, that does not have the .prj anciliary file (the example is equally good for any case where the prj file contents cannot be matched to an official EPSG code).

We are going to post the following import definition:

```
{
   "import": {
      "targetWorkspace": {
         "workspace": {
            "name": "tasmania"
         }
      },
      "data": {
        "type": "file",
        "file": "C:/data/tasmania/tasmania_cities.shp"
      }
   }
}
```

With the usual curl command:

```
curl -u admin:geoserver -XPOST -H "Content-type: application/json" -d @import.json
↪"http://localhost:8080/geoserver/rest/imports"
```

The response in case the CRS is missing will be:

```
{
  "import": {
    "id": 13,
    "href": "http://localhost:8080/geoserver/rest/imports/13",
    "state": "PENDING",
    "archive": false,
    "targetWorkspace": {
      "workspace": {
        "name": "tasmania"
      }
    },
    "data": {
      "type": "file",
      "format": "Shapefile",
      "file": "tasmania_cities.shp"
    },
    "tasks": [
      {
        "id": 0,
        "href": "http://localhost:8080/geoserver/rest/imports/13/tasks/0",
        "state": "NO_CRS"
```

```
      }
    ]
  }
}
```

Drilling into the task layer we can see the srs information is missing:

```
{
  "layer": {
    "name": "tasmania_cities",
    "href": "http://localhost:8080/geoserver/rest/imports/13/tasks/0/layer",
    "title": "tasmania_cities",
    "originalName": "tasmania_cities",
    "nativeName": "tasmania_cities",
    "bbox": {
      "minx": 146.2910004483,
      "miny": -43.85100181689,
      "maxx": 148.2910004483,
      "maxy": -41.85100181689
    },
    "attributes": [
      {
        "name": "the_geom",
        "binding": "org.locationtech.jts.geom.MultiPoint"
      },
      {
        "name": "CITY_NAME",
        "binding": "java.lang.String"
      },
      {
        "name": "ADMIN_NAME",
        "binding": "java.lang.String"
      },
      {
        "name": "CNTRY_NAME",
        "binding": "java.lang.String"
      },
      {
        "name": "STATUS",
        "binding": "java.lang.String"
      },
      {
        "name": "POP_CLASS",
        "binding": "java.lang.String"
      }
    ],
    "style": {
      "name": "tasmania_tasmania_cities2",
      "href": "http://localhost:8080/geoserver/rest/imports/13/tasks/0/layer/style"
    }
  }
}
```

The following PUT request will update the SRS:

```
curl -u admin:geoserver -XPUT -H "Content-type: application/json" -d @layerUpdate.
↪json "http://localhost:8080/geoserver/rest/imports/13/tasks/0/layer/"
```

Where `layerUpdate.json` is:

```
{
   layer : {
      srs: "EPSG:4326"
   }
}
```

Getting the import definition again, we'll find it ready to execute:

```
{
  "import": {
    "id": 13,
    "href": "http://localhost:8080/geoserver/rest/imports/13",
    "state": "PENDING",
    "archive": false,
    "targetWorkspace": {
      "workspace": {
        "name": "tasmania"
      }
    },
    "data": {
      "type": "file",
      "format": "Shapefile",
      "file": "tasmania_cities.shp"
    },
    "tasks": [
      {
        "id": 0,
        "href": "http://localhost:8080/geoserver/rest/imports/13/tasks/0",
        "state": "READY"
      }
    ]
  }
}
```

A POST request will make it execute:

```
curl -u admin:geoserver -XPOST "http://localhost:8080/geoserver/rest/imports/13"
```

And eventually succeed:

```
{
  "import": {
    "id": 13,
    "href": "http://localhost:8080/geoserver/rest/imports/13",
    "state": "COMPLETE",
    "archive": false,
    "targetWorkspace": {
      "workspace": {
        "name": "tasmania"
      }
    },
    "data": {
      "type": "file",
      "format": "Shapefile",
      "file": "tasmania_cities.shp"
    },
    "tasks": [
```

```
      {
        "id": 0,
        "href": "http://localhost:8080/geoserver/rest/imports/13/tasks/0",
        "state": "COMPLETE"
      }
    ]
  }
}
```

Uploading a Shapefile to PostGIS

This example shows the process for uploading a Shapefile (in a zip file) to an existing PostGIS datastore (cite:postgis).

Create the import definition:

```
{
  "import": {
    "targetStore": {
      "dataStore": {
        "name": "postgis"
      }
    },
    "targetWorkspace": {
      "workspace": {
        "name": "cite"
      }
    }
  }
}
```

POST this definition to /geoserver/rest/imports:

```
curl -u admin:geoserver -XPOST -H "Content-type: application/json" -d @import.json
↪"http://localhost:8080/geoserver/rest/imports"
```

The response will contain the import ID.

We now have an empty import with no tasks. To add a task, POST the shapefile to the list of tasks:

```
curl -u admin:geoserver -F name=myshapefile.zip -F filedata=@myshapefile.zip "http://
↪localhost:8080/geoserver/rest/imports/14/tasks"
```

Since we sent a shapefile, importer assumes the target will be a shapefile store. To import to PostGIS, we will need to reset it. Create the following JSON file:

```
{
  "dataStore": {
    "name":"postgis"
  }
}
```

PUT this file to /geoserver/rest/imports/14/tasks/0/target:

```
curl -u admin:geoserver -XPUT -H "Content-type: application/json" -d @target.json
↪"http://localhost:8080/geoserver/rest/imports/14/tasks/0/target"
```

Finally, we execute the import by sending a POST to /geoserver/rest/imports/14:

```
curl -u admin:geoserver -XPOST "http://localhost:8080/geoserver/rest/imports/14"
```

Uploading a CSV file to PostGIS while transforming it

A remote sensing tool is generating CSV files with some locations and measurements, that we want to upload into PostGIS as a new spatial table. The CSV file looks as follows:

```
AssetID, SampleTime, Lat, Lon, Value
1, 2015-01-01T10:00:00, 10.00, 62.00, 15.2
1, 2015-01-01T11:00:00, 10.10, 62.11, 30.25
1, 2015-01-01T12:00:00, 10.20, 62.22, 41.2
1, 2015-01-01T13:00:00, 10.31, 62.33, 27.6
1, 2015-01-01T14:00:00, 10.41, 62.45, 12
```

First, we are going to create a empty import with an existing postgis store as the target:

```
curl -u admin:geoserver -XPOST -H "Content-type: application/json" -d @import.json
↪"http://localhost:8080/geoserver/rest/imports"
```

Where import.json is:

```
{
   "import": {
      "targetWorkspace": {
         "workspace": {
            "name": "topp"
         }
      },
      "targetStore": {
         "dataStore": {
            "name": "gttest"
         }
      }
   }
}
```

Then, we are going to POST the csv file to the tasks list, in order to create an import task for it:

```
curl -u admin:geoserver -F name=test -F filedata=@values.csv "http://localhost:8080/
↪geoserver/rest/imports/0/tasks"
```

And we are going to get back a new task definition, with a notification that the CRS is missing:

```
{
  "task": {
    "id": 0,
    "href": "http://localhost:8080/geoserver/rest/imports/16/tasks/0",
    "state": "NO_CRS",
    "updateMode": "CREATE",
    "data": {
      "type": "file",
      "format": "CSV",
      "file": "values.csv"
    },
    "target": {
```

```
      "href": "http://localhost:8080/geoserver/rest/imports/16/tasks/0/target",
      "dataStore": {
        "name": "values",
        "type": "CSV"
      }
    },
    "progress": "http://localhost:8080/geoserver/rest/imports/16/tasks/0/progress",
    "layer": {
      "name": "values",
      "href": "http://localhost:8080/geoserver/rest/imports/16/tasks/0/layer"
    },
    "transformChain": {
      "type": "vector",
      "transforms": [

      ]
    }
  }
}
```

As before, we are going to force the CRS by updating the layer:

```
curl -u admin:geoserver -XPUT -H "Content-type: application/json" -d @layerUpdate.
↪json "http://localhost:8080/geoserver/rest/imports/0/tasks/0/layer/"
```

Where `layerUpdate.json` is:

```
{
   layer : {
      srs: "EPSG:4326"
   }
}
```

Then, we are going to create a transformation mapping the Lat/Lon columns to a point:

```
{
  "type": "AttributesToPointGeometryTransform",
  "latField": "Lat",
  "lngField": "Lon"
}
```

The above will be uploaded to GeoServer as follows:

```
curl -u admin:geoserver -XPOST -H "Content-type: application/json" -d @toPoint.json
↪"http://localhost:8080/geoserver/rest/imports/0/tasks/0/transforms"
```

Now the import is ready to run, and we'll execute it using:

```
curl -u admin:geoserver -XPOST "http://localhost:8080/geoserver/rest/imports/0"
```

If all goes well the new layer is created in PostGIS and registered in GeoServer as a new layer.

In case the features in the CSV need to be appended to an existing layer a PUT request against the task might be performed, changing its updateMode from "CREATE" to "APPEND". Changing it to "REPLACE" instead will preserve the layer, but remove the old conents and replace them with the newly uploaded ones.

Uploading and optimizing a GeoTiff with ground control points

A data supplier is periodically providing GeoTiffs that we need to configure in GeoServer. The GeoTIFF is referenced via Ground Control Points, is organized by stripes, and has no overviews. The objective is to rectify, optimize and publish it via the importer.

First, we are going to create a empty import with no store as the target:

```
curl -u admin:geoserver -XPOST -H "Content-type: application/json" -d @import.json
↪"http://localhost:8080/geoserver/rest/imports"
```

Where import.json is:

```
{
   "import": {
      "targetWorkspace": {
         "workspace": {
            "name": "sf"
         }
      }
   }
}
```

Then, we are going to POST the GeoTiff file to the tasks list, in order to create an import task for it:

```
curl -u admin:geoserver -F name=test -F filedata=@box_gcp_fixed.tif "http://
↪localhost:8080/geoserver/rest/imports/0/tasks"
```

We are then going to append the transformations to rectify (gdalwarp), retile (gdal_translate) and add overviews (gdaladdo) to it:

```
curl -u admin:geoserver -XPOST -H "Content-type: application/json" -d @warp.json
↪"http://localhost:8080/geoserver/rest/imports/0/tasks/0/transforms"
curl -u admin:geoserver -XPOST -H "Content-type: application/json" -d @gtx.json
↪"http://localhost:8080/geoserver/rest/imports/0/tasks/0/transforms"
curl -u admin:geoserver -XPOST -H "Content-type: application/json" -d @gad.json
↪"http://localhost:8080/geoserver/rest/imports/0/tasks/0/transforms"
```

`warp.json` is:

```
{
  "type": "GdalWarpTransform",
  "options": [ "-t_srs", "EPSG:4326"]
}
```

`gtx.json` is:

```
{
  "type": "GdalTranslateTransform",
  "options": [ "-co", "TILED=YES", "-co", "BLOCKXSIZE=512", "-co", "BLOCKYSIZE=512"]
}
```

`gad.json` is:

```
{
  "type": "GdalAddoTransform",
  "options": [ "-r", "average"],
```

```
  "levels" : [2, 4, 8, 16]
}
```

Now the import is ready to run, and we'll execute it using:

```
curl -u admin:geoserver -XPOST "http://localhost:8080/geoserver/rest/imports/0"
```

A new layer `box_gcp_fixed` layer will appear in GeoServer, with an underlying GeoTiff file ready for web serving.

Adding a new granule into an existing mosaic

A data supplier is periodically providing new time based imagery that we need to add into an existing mosaic in GeoServer. The imagery is in GeoTiff format, and lacks a good internal structure, which needs to be aligned with the one into the other images.

First, we are going to create a import with an indication of where the granule is located, and the target store:

> curl -u admin:geoserver -XPOST -H "Content-type: application/json" -d @import.json "http://localhost:8080/geoserver/rest/imports"

Where import.json is:

```
{
   "import": {
      "targetWorkspace": {
         "workspace": {
            "name": "topp"
         }
      },
      "data": {
        "type": "file",
        "file": "/home/aaime/devel/gisData/ndimensional/data/world/world.200407.
↪3x5400x2700.tiff"
      },
      "targetStore": {
         "dataStore": {
            "name": "bluemarble"
         }
      }
   }
}
```

We are then going to append the transformations to harmonize the file with the rest of the mosaic:

```
curl -u admin:geoserver -XPOST -H "Content-type: application/json" -d @gtx.json
↪"http://localhost:8080/geoserver/rest/imports/0/tasks/0/transforms"
curl -u admin:geoserver -XPOST -H "Content-type: application/json" -d @gad.json
↪"http://localhost:8080/geoserver/rest/imports/0/tasks/0/transforms"
```

`gtx.json` is:

```
{
  "type": "GdalTranslateTransform",
  "options": [ "-co", "TILED=YES"]
}
```

`gad.json` is:

```
{
  "type": "GdalAddoTransform",
  "options": [ "-r", "average"],
  "levels" : [2, 4, 8, 16, 32, 64, 128]
}
```

Now the import is ready to run, and we'll execute it using:

```
curl -u admin:geoserver -XPOST "http://localhost:8080/geoserver/rest/imports/0"
```

The new granule will be ingested into the mosaic, and will thus be available for time based requests.

Asynchronously fetching and importing data from a remote server

We assume a remote FTP server contains multiple shapefiles that we need to import in GeoServer as new layers. The files are large, and the server has much better bandwith than the client, so it's best if GeoServer performs the data fetching on its own.

In this case a asynchronous request using `remote` data will be the best fit:

```
curl -u admin:geoserver -XPOST -H "Content-type: application/json" -d @import.json
↪"http://localhost:8080/geoserver/rest/imports?async=true"
```

Where import.json is:

```
{
   "import": {
      "targetWorkspace": {
         "workspace": {
            "name": "topp"
         }
      },
      "data": {
        "type": "remote",
        "location": "ftp://myserver/data/bc_shapefiles",
        "username": "dan",
        "password": "secret"
      }
   }
}
```

The request will return immediately with an import context in "INIT" state, and it will remain in such state until the data is fetched and the tasks created. Once the state switches to "PENDING" the import will be ready for execution. Since there is a lot of shapefiles to process, also the import run will be done in asynchronous mode:

```
curl -u admin:geoserver -XPOST "http://localhost:8080/geoserver/rest/imports/0?
↪async=true"
```

The response will return immediately in this case as well, and the progress can be followed as the tasks in the import switch state.

Importing and optimizing a large image with a single request

A large image appears every now and then on a mounted disk share, the image needs to be optimized and imported into GeoServer as a new layer. Since the source is large and we need to copy it on the local disk where the data dir resides, a "remote" data is the right tool for the job, an asynchronous execution is also recommended to avoid waiting on a possibly large command. In this case the request will also contains the "exec=true" parameter to force the importer an immediate execution of the command.

The request will then look as follows:

```
curl -u admin:geoserver -XPOST -H "Content-type: application/json" -d @import.json
↪"http://localhost:8080/geoserver/rest/imports?async=true&exec=true"
```

Where import.json is:

```
{
  "import": {
    "targetWorkspace": {
      "workspace": {
        "name": "topp"
      }
    },
    "data": {
      "type": "remote",
      "location": "\/mnt\/remoteDisk\/bluemarble.tiff"
    },
    "transforms": [
      {
        "type": "GdalTranslateTransform",
        "options": [
          "-co", "TILED=YES",
          "-co", "COMPRESS=JPEG",
          "-co", "JPEG_QUALITY=85",
          "-co", "PHOTOMETRIC=YCBCR"
        ]
      },
      {
        "type": "GdalAddoTransform",
        "options": [
          "-r",
          "average",
          "--config", "COMPRESS_OVERVIEW", "JPEG",
          "--config", "PHOTOMETRIC_OVERVIEW", "YCBCR"
        ],
        "levels": [ 2, 4, 8, 16, 32, 64 ]
      }
    ]
  }
}
```

Given the request is asynchronous, the client will have to poll the server in order to check if the initialization and execution have succeded.

15.7 INSPIRE

The INSPIRE extension allows GeoServer to be compliant with the View Service and Download Service specifications put forth by the **Infrastructure for Spatial Information in the European Community** (INSPIRE) directive.

In practice this means adding some extra elements into an extended capabilities section of the WMS, WFS and WCS capabilities documents. For WMS, WFS and WCS this includes a **Metadata URL** element with a link to the metadata associated with the service, and **SupportedLanguages** and **ResponseLanguage** elements which report the response language (GeoServer can only support one response language). For WFS and WCS there are also one or more **SpatialDataSetIdentifier** elements for each spatial data resource served by the service.

Note: The current INSPIRE extension fulfills "Scenario 1" of the View Service extended metadata requirements. "Scenario 2" is not currently supported in GeoServer, but is certainly possible to implement. If you are interested in implementing or funding this, please raise the issue on the *GeoServer mailing list*.

For more information on the INSPIRE directive, please see the European Commission's INSPIRE website.

15.7.1 Installing the INSPIRE extension

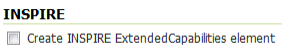
The INSPIRE extension is a official extension available at GeoServer download pages (starting with GeoServer 2.3.2).

1. Download the inspire zip release file from the download page of your version of GeoServer
2. Extract the archive and copy the contents into the `<GEOSERVER_ROOT>/WEB-INF/lib` directory.
3. Restart GeoServer.

To verify that the extension was installed successfully, please see the next section on *Using the INSPIRE extension*.

15.7.2 Using the INSPIRE extension

When the INSPIRE extension has been properly installed, the *WMS settings*, *WFS settings* and *WCS settings* sections of the *Web administration interface* will show an extra INSPIRE configuration section. If the data directory has not been configured with INSPIRE parameters before, this section will just contain a check box to enable the creation of an INSPIRE ExtendedCapabilities element.



Note: If you do not see this content in the service configuration pages, the INSPIRE extension may not be installed properly. Reread the section on *Installing the INSPIRE extension* and verify that the correct file was saved to the correct directory.

Extended WMS and WMTS configuration

INSPIRE-specific configuration is accessed on the main *WMS settings* or WMTS settings page in the *Web administration interface*. This is accessed by clicking on the *WMS* or *WMTS* link on the sidebar.

Note: You must be logged in as an administrator to edit WMS or WMTS configuration.

Once on the service configuration page, there will be a block titled *INSPIRE*. If you enable the checkbox shown above this section will have three additional settings:

- *Language* combo box, for setting the Supported, Default, and Response languages
- *Service Metadata URL* field, a URL containing the location of the metadata associated with the service
- *Service Metadata Type* combo box, for detailing whether the metadata came from a CSW (Catalog Service) or a standalone metadata file

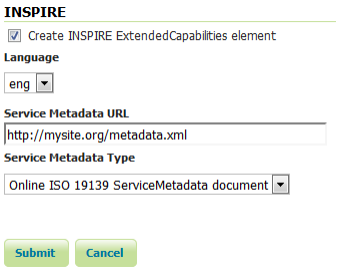


Fig. 15.24: *INSPIRE-related options*

After clicking *Submit* on this page, any changes will be immediately reflected in the services (WMS 1.3.0 or WMTS 1.0.0) capabilities document.

Note: At the time of writing the INSPIRE Schemas only allow 23 choices for *Language*. The GeoServer INSPIRE extension allows some other languages to be chosen. If you choose one of these your capabilities document won't be Schema valid but, as discussed in issue 7388, the INSPIRE Schemas seem to be at fault. If you have some other language you want adding to the list then please *raise the issue*.

Note: The *Service Metadata URL* field is mandatory so you will not be allowed to submit a blank value.

Note: The *Service Metadata Type* combo box only allows to select the appropriate MIME type for a CSW response or standalone metadata file or to omit a value altogether. If you think other values would be useful you could raise the issue on the *GeoServer mailing list*. In the meantime it is possible to manually edit the created configuration files as a workaround.

Note: Currently GeoServer does not offer the ability to configure alternate languages, as there is no way for an administrator to configure multiple responses. There is an open issue on the GeoServer issue tracker that we are hoping to secure funding for. If you are interested in implementing or funding this improvement, please raise the issue on the *GeoServer mailing list*.

Extended WMS and WMTS Capabilities

Note: The INSPIRE extension only modifies the WMS 1.3.0 response, so please make sure that you are viewing the correct capabilities document.

The WMS 1.3.0 and WMTS 1.0.0 capabilities document will contain two additional entries in the `xsi:schemaLocation` of the root `<WMS_Capabilities>` tag once the INSPIRE extension is installed:

- `http://inspire.ec.europa.eu/schemas/inspire_vs/1.0`
- `http://inspire.ec.europa.eu/schemas/inspire_vs/1.0/inspire_vs.xsd`

If you have enabled the check box to create the INSPIRE ExtendedCapabilities element and entered the values described in the previous section then there will also be an additional ExtendedCapabilities block. This tag block shows up in between the tags for `<Exception>` and `<Layer>`. It contains the following information:

- Metadata URL and MIME type
- Supported Language(s)
- Response Language

With the example values shown in the above configuration panel, this block would contain the following content:

```
<inspire_vs:ExtendedCapabilities>
 <inspire_common:MetadataUrl>
  <inspire_common:URL>http://mysite.org/metadata.xml</inspire_common:URL>
  <inspire_common:MediaType>
   application/vnd.iso.19139+xml
  </inspire_common:MediaType>
 </inspire_common:MetadataUrl>
 <inspire_common:SupportedLanguages>
  <inspire_common:DefaultLanguage>
   <inspire_common:Language>eng</inspire_common:Language>
  </inspire_common:DefaultLanguage>
 </inspire_common:SupportedLanguages>
 <inspire_common:ResponseLanguage>
  <inspire_common:Language>eng</inspire_common:Language>
 </inspire_common:ResponseLanguage>
</inspire_vs:ExtendedCapabilities>
```

ISNPIRE recommends that every layer offered by a INSPIRE WMTS should use the InspireCRS84Quad grid set which is already configured in GeoServer, but is up to the user to select it when publishing a INSPIRE WMTS layer.

Extended WFS and WCS configuration

INSPIRE-specific configuration is accessed on the main *WFS settings* and *WCS settings* pages in the *Web administration interface*. These are accessed by clicking on the *WFS* and *WCS* links on the sidebar respectively.

Note: You must be logged in as an administrator to edit WFS configuration.

Once on the WFS or WCS configuration page, there will be a block titled *INSPIRE*. If you enable the check-box shown above this section will have the following additional settings:

- *Language* combo box, for setting the Supported, Default, and Response languages

- *Service Metadata URL* field, a URL containing the location of the metadata associated with the WFS or WCS
- *Service Metadata Type* combo box, for detailing whether the metadata came from a CSW (Catalog Service) or a standalone metadata file
- *Spatial dataset identifers* table, where you can specify a code (mandatory), a namespace (optional) and a metadata URL (optional) for each spatial data set the WFS or WCS is offering

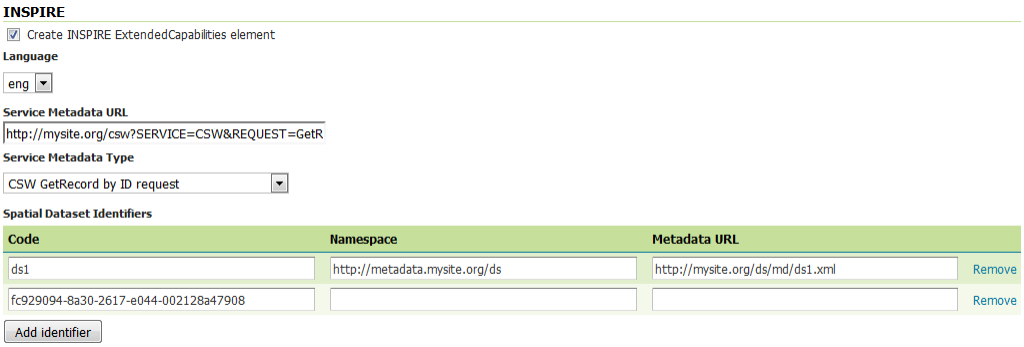


Fig. 15.25: *INSPIRE-related options*

After clicking *Submit* on this page, any changes will be immediately reflected in the WFS 1.1 and WFS 2.0 or WCS 2.0 capabilities documents as appropriate.

Note: At the time of writing the INSPIRE Schemas only allow 23 choices for *Language*. The GeoServer INSPIRE extension allows some other languages to be chosen. If you choose one of these your capabilities document won't be Schema valid but, as discussed in issue 7388, the INSPIRE Schemas seem to be at fault. If you have some other language you want adding to the list then please *raise the issue*.

Note: The *Service Metadata URL* field and at least one *Spatial dataset identifers* entry are mandatory so you will not be allowed to submit the page without these.

Note: The *Service Metadata Type* combo box only allows to select the appropriate MIME type for a CSW response or standalone metadata file or to omit a value altogether. If you think other values would be useful you could raise the issue on the *GeoServer mailing list*. In the meantime it is possible to manually edit the created configuration files as a workaround.

Note: Currently GeoServer does not offer the ability to configure alternate languages, as there is no way for an administrator to configure multiple responses. There is an open issue on the GeoServer issue tracker that we are hoping to secure funding for. If you are interested in implementing or funding this improvement, please raise the issue on the *GeoServer mailing list*.

Extended WFS and WCS Capabilities

Note: The INSPIRE directive is relevant to WFS 1.1 and 2.0 and WCS 2.0 only, so please make sure that you are viewing the correct capabilities document.

The WFS and WCS capabilities documents will contain two additional entries in the `xsi:schemaLocation` of the root element tag once the INSPIRE extension is installed:

- `http://inspire.ec.europa.eu/schemas/common/1.0/common.xsd`
- `http://inspire.ec.europa.eu/schemas/inspire_dls/1.0/inspire_dls.xsd`

If you have enabled the check box to create the INSPIRE ExtendedCapabilities element and entered the values described in the previous section then there will also be an additional ExtendedCapabilities block with the following information:

- Metadata URL and MIME type
- Supported Language(s)
- Response Language
- Spatial data identifier(s)

With the example values shown in the above configuration panel, this block would contain the following content:

```
<inspire_dls:ExtendedCapabilities>
 <inspire_common:MetadataUrl>
  <inspire_common:URL>
   http://mysite.org/csw?SERVICE=CSW&REQUEST=GetRecordById&ID=wfs2&
  </inspire_common:URL>
  <inspire_common:MediaType>
   application/vnd.ogc.csw.GetRecordByIdResponse_xml
  </inspire_common:MediaType>
 </inspire_common:MetadataUrl>
 <inspire_common:SupportedLanguages>
  <inspire_common:DefaultLanguage>
   <inspire_common:Language>eng</inspire_common:Language>
  </inspire_common:DefaultLanguage>
 </inspire_common:SupportedLanguages>
 <inspire_common:ResponseLanguage>
  <inspire_common:Language>eng</inspire_common:Language>
 </inspire_common:ResponseLanguage>
 <inspire_dls:SpatialDataSetIdentifier
  metadataURL="http://mysite.org/ds/md/ds1.xml">
  <inspire_common:Code>ds1</inspire_common:Code>
  <inspire_common:Namespace>
   http://metadata.mysite.org/ds
  </inspire_common:Namespace>
 </inspire_dls:SpatialDataSetIdentifier>
 <inspire_dls:SpatialDataSetIdentifier>
  <inspire_common:Code>
   fc929094-8a30-2617-e044-002128a47908
  </inspire_common:Code>
 </inspire_dls:SpatialDataSetIdentifier>
</inspire_dls:ExtendedCapabilities>
```

The spatial data identifiers section is mandatory, but cannot be filled by default, it is your duty to provide at least one spatial dataset identifier (see the INSPIRE download service technical guidelines for more information).

15.8 JP2K Plugin

GeoServer can leverage the JP2K Geotools plugin to read JP2K coverage formats. In case you have a Kakadu license and you have built your set of native libraries, you will be able to access the JP2K data with higher performances leveraging on it. Otherwise you will use the standard SUN's JP2K. See http://docs.geotools.org/latest/userguide/library/coverage/jp2k.html for further information.

15.8.1 Installing Kakadu

In order for GeoServer to leverage on the Kakadu libraries, the Kakadu binaries must be installed through your host system's OS.

If you are on Windows, make sure that the Kakadu DLL files are on your PATH. If you are on Linux, be sure to set the LD_LIBRARY_PATH environment variable to be the folder where the SOs are extracted.

Once these steps have been completed, restart GeoServer. If done correctly, new data formats will be in the Raster Data Sources list when creating a new data store:

Fig. 15.26: *Raster Data Source*

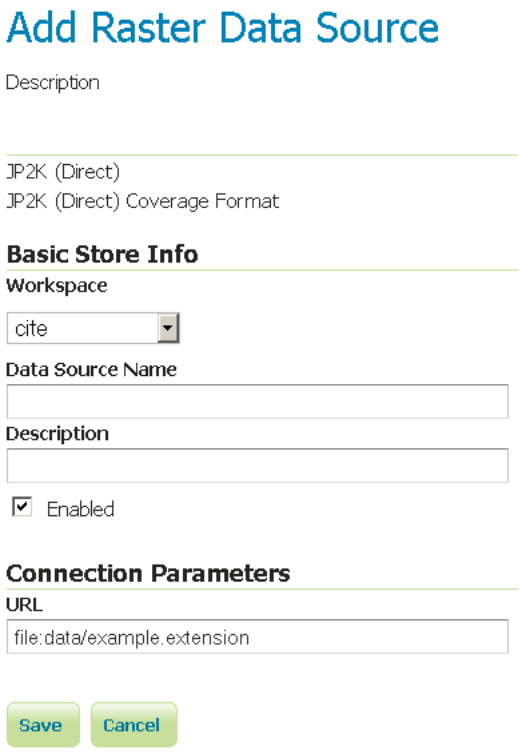


Fig. 15.27: *Configuring a JP2K data store*

15.9 libjpeg-turbo Map Encoder Extension

This plugin brings in the ability to encode JPEG images as WMS output using the libjpeg-turbo library. Citing its website the libjpeg-turbo library is a derivative of libjpeg that uses SIMD instructions (MMX, SSE2, NEON) to accelerate baseline JPEG compression and decompression on x86, x86-64, and ARM systems. On such systems, libjpeg-turbo is generally 2-4x as fast as the unmodified version of libjpeg, all else being equal. I guess it is pretty clear why we wrote this plugin! Note that the underlying imageio-ext-turbojpeg uses TurboJpeg which is a higher level set of API (providing more user-friendly methods like "Compress") built on top of libjpeg-turbo.

Warning: The speedup may vary depending on the target infrastructure.

The module, once installed, simply replace the standard JPEG encoder for GeoServer and allows us to use the libjpeg-turbo library to encode JPEG response for GetMap requests.

Note: It is worth to point out that the module depends on a successful installation of the libjpeg-turbo native libraries (more on this later).

15.9.1 Installing the libjpeg-turbo native library

Installing the libjpeg-turbo native library is a precondition to have the relative GeoServer Map Encoder properly installed; once the GeoServer extension has been installed as we explain in the following section, the needed JARs with the Java bridge to the library are in the classpath, therefore all we need to do is to install the native library itself to start encoding JPEG at turbo speed.

To perform the installation of the libjpeg-turbo binaries (or native library) you have to perform the following steps:

1. go to the download site here and download the latest available stable release (1.2.90 at the time of writing)
2. select the package that matches the target platform in terms of Operating System (e.g. Linux rather than Windows) and Architecture (32 vs 64 bits)
3. perform the installation using the target platform conventions. As an instance for Windows you should be using an installer that installs all the needed libs in a location at user's choice. On Ubuntu Linux systems you can use the *deb* files insted.
4. Once the native libraries are installed, you have to make sure the GeoServer can load them. This should happen automatically after Step 2 on Linux, while on Windows you should make sure that the location where you placed the DLLs is part of the PATH environment variable for the Java Process for the GeoServer.

Warning: When installing on Windows, always make sure that the location where you placed the DLLs is part of the PATH environment variable for the Java Process for the GeoServer. This usually means that you have to add such location to the PATH environmental variable for the user that is used to run GeoServer or the system wide variables.

Warning: When installing on Linux, make sure that the location where you placed the DLLs is part of the LD_LIBRARY_PATH environment variable for the Java Process for the GeoServer. This usually happens automatically for the various Linux packages, but in some cases you might be forced to do that manually

Note: It does not hurt to add also the location where where the native libraries where installed to the Java startup options -Djava.library.path=<absolute_and_valid_path>

15.9.2 Installing the GeoServer libjpeg-turbo extension

Warning: Before moving on make sure you installed the libjpeg-turbo binaries as per the section above.

1. Download the extension from the nightly GeoServer extensions builds.

   Warning: Make sure to match the version of the extension to the version of the GeoServer instance!

2. Extract the contents of the archive into the `WEB-INF/lib` directory of the GeoServer installation.

15.9.3 Checking if the extension is enabled

Once the extension is installed, the following lines should appear in the GeoServer log:

```
10-mar-2013 19.16.28 it.geosolutions.imageio.plugins.turbojpeg.TurboJpegUtilities load
INFO: TurboJPEG library loaded (turbojpeg)
```

or:

```
10 mar 19:17:12 WARN [turbojpeg.TurboJPEGMapResponse] - The turbo jpeg encoder is␣
↪available for usage
```

15.9.4 Disabling the extension

When running GeoServer the turb encoder can be disabled by using the Java switch for the JVM process:

```
-Ddisable.turbojpeg=true
```

In this case a message like the following should be found in the log:

```
WARN [map.turbojpeg] - The turbo jpeg encoder has been explicitly disabled
```

Note: We will soon add a section in the GUI to check the status of the extension and to allow users to enable/disable it at runtime.

15.10 Monitoring

The monitor extension tracks requests made against a GeoServer instance. With the extension request data can be persisted to a database, used to generate simple reports , and routed to a customized request audit log.

To get the extension proceed to *Installing the Monitor Extension*. To learn more about how it works jump to the *Monitoring Overview* section.

15.10.1 Installing the Monitor Extension

Note: If performing an upgrade of the monitor extension please see *Upgrading*.

The monitor extension is not part of the GeoServer core and must be installed as a plug-in. To install:

1. Navigate to the GeoServer download page.
2. Find the page that matches the version of the running GeoServer.
3. Download the monitor extension. The download link will be in the *Extensions* section under *Other*.
4. Extract the files in this archive to the `WEB-INF/lib` directory of your GeoServer installation.
5. Restart GeoServer

Verifying the Installation

There are two ways to verify that the monitoring extension has been properly installed.

1. Start GeoServer and open the *Web administration interface*. Log in using the administration account. If successfully installed, there will be a *Monitor* section on the left column of the home page.

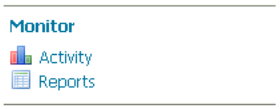


Fig. 15.28: *Monitoring section in the web admin interface*

1. Start GeoServer and navigate to the current *GeoServer data directory*. If successfully installed, a new directory named `monitoring` will be created in the data directory.

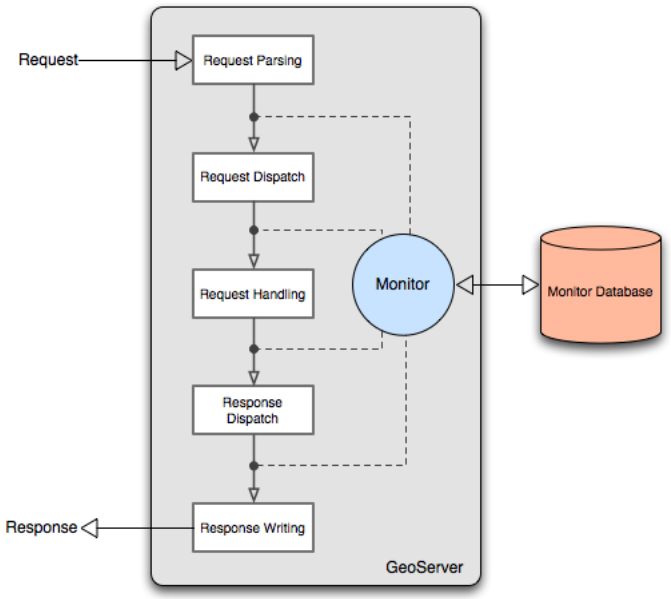


Fig. 15.29: *Monitor extension architecture*

15.10.2 Monitoring Overview

The following diagram outlines the architecture of the monitor extension:

As a request is processed the monitor inserts itself at particular points in the request life cycle to capture various information about the request. Such information includes:

- Timestamp of the origin of the request
- Total time it took for the request to complete
- Origin of the request
- HTTP information such as the body content type, header information, etc...

And more. See the *Data Reference* section for a complete list.

In addition to capturing request data the monitor extension is also capable of persisting it. Two options are provided out of the box:

- Persisting to a relational database, see *Database Persistence* for more details
- Piping to a log file, see *Audit Logging* for more details

By default the extension will do neither and simply maintain data for only the most recent requests. The data is stored in memory meaning that if the server is restarted or shutdown this information is lost.The *Monitor Configuration* section provides a comprehensive guide to configuring the monitor extension.

Stored request information is made available through a simple *query api* that allows clients to access request data through a HTTP interface.

15.10.3 Data Reference

The following is a list of all the attributes of a request that are captured by the monitor extension.

General

Attribute	Description	Type
ID	Numeric identifier of the request. Every request is assigned an identifier upon its creation.	Numeric
Status	Status of the request. See *notes* below.	String
Category	The type of request being made, for example an OGC service request, a REST call, etc... See *notes* below.	String
Start time	The time of the start of the request.	Timestamp
End time	The time of the completion of the request.	Timestamp
Total time	The total time spent handling the request, measured in milliseconds, equal to the end time - start time.	Numeric
Error message	The exception message if the request failed or resulted in an error.	String
Error	The raw exception if the message failed or resulted in an error.	Text blob

Status

The status of a request changes over it's life cycle and may have one of the following values:

- `WAITING` - The request has been received by the server, but is queued and not yet being actively handled.
- `RUNNING` - The request is in the process of being handled by the server.
- `FINISHED` - The request has been completed and finished normally.
- `FAILED` - The request has been completed but resulted in an error.
- `CANCELLED` - The request was cancelled before it could complete.
- `INTERRUPTED` - The request was interrupted before it could complete.

Category

Requests are grouped into categories that describe the nature or type of the request. The following are the list of all categories:

- `OWS` - The request is an OGC service request.
- `REST` - The request is a REST service request.
- `OTHER` - All other requests.

HTTP

The following attributes are all HTTP related.

Attribute	Description	Type
HTTP method	The HTTP method, one of `GET`, `POST`, `PUT`, or `DELETE`	String
Remote address	The IP address of the client from which the request originated.	String
Remote host	The hostname corresponding to the remote address, obtained via reverse DNS lookup.	String
Host	The hostname of the server handling the request, from the point of view of the client.	String
Internal host	The hostname of the server handling request, from the point of view of the local network. Availability depends on host and network configuration.	String
Path	The path component of the request URL, for example: "/wms", "/rest/workspaces.xml", etc...	String
Query string	The query string component of the request URL. Typically only present when the HTTP method is GET.	String
Body	The body content of the request. Typically only present when the HTTP method is PUT or POST.	Binary blob
Body content length	The total number of bytes comprising the body of the request. Typically only present when the HTTP method is PUT or POST.	Numeric
Body content type	The mime type of the body content of the request, for example: "application/json", "text/xml; subtype=gml/3.2", etc... Typically only present when the HTTP method is PUT or POST.	String
Response status	The HTTP response code, for example: 200, 401, etc...	Numeric
Response length	The total number of bytes comprising the response to the request.	Numeric
Response content type	The mime type of the response to the request.	String
Remote user	The username specified parsed of the request. Only available when request included credentials for authentication.	String
Remote user agent	The value of the `User-Agent` HTTP header.	String
Http referrer	The value of the `Referer` HTTP header.	String

OWS/OGC

The following attributes are OGC service specific.

Attribute	Description	Type
Service	The OGC service identifier, for example: "WMS", "WFS", etc...	String
Operation	The OGC operation name, for example: "GetMap", "GetFeature", etc...	String
Sub operation	The ogc sub operation (if it applies). For instance when the operation is a WFS Transaction the sub operation may be one of "Insert", "Update", etc...	String
OWS/OGC Version	The OGC service version, for example with WFS the version may be "1.0.0", "1.1.0", etc...	String
Resources	Names of resources (layers, processes, etc...) specified as part of the request.	List of String
Bounding box	The bounding box specified as part of the request. In some cases this is not possible to obtain this reliable, an example being a complex WFS query with a nested "BBOX" filter.	List of Numeric

GeoIP

The following attributes are specific to GeoIP look ups and are not captured out of the box. See *GeoIP* for more details.

Attribute	Description	Type
Remote country	Name of the country of the client from which the request originated.	String
Remote city	Name of the city from which the request originated.	String
Remote lat	The latitude from which the request originated.	Numeric
Remote lon	The longitude from which the request originated.	Numeric

GWC

The following attributes are specific to tile cached requests.

Attribute	Description	Type
CacheResult	"HIT" or "MISS" (can be empty if GWC was not involved)	String
MissReason	A description of why the cache was not used. Available only on requests hitting a cached layer on direct WMS integration, applies to cases where the request was not forwarded to GWC, for example "no parameter filter exists for FEATUREID", "request does not align to grid(s) "EPSG:4326" or "not a tile layer". Will be missing for any request not hitting the direct integration (e.g., direct WMTS requests, for example)	String

15.10.4 Monitor Configuration

Many aspects of the monitor extension are configurable. All configuration files are stored in the data directory under the `monitoring` directory:

```
<data_directory>
    monitoring/
        filter.properties
        monitor.properties
```

The `monitor.properties` file is the main configuration file whose contents are described in the following sections. The `filter.properties` allows for *filtering* out those requests from being monitored.

Database persistence with hibernate is described in more detail in the *Database Persistence* section.

Monitor Storage

How request data is persisted is configurable via the `storage` property defined in the `monitor.properties` file. The following values are supported for the `storage` property:

- **memory** - Request data is to be persisted in memory alone.

The default value is `memory`.

The "monitor hibernate" community module allows to also store the requests in a relational database.

Memory Storage

With memory storage only the most recent 100 requests are stored. And by definition this storage is volatile in that if the GeoServer instance is restarted, shutdown, or crashes this data is lost.

Monitor Mode

The monitor extension supports different "monitoring modes" that control how request data is captured. Currently two modes are supported:

- **history** *(Default)* - Request information updated post request only. No live information made available.
- **live** - Information about a request is captured and updated in real time.

The monitor mode is set with the `mode` property in the `monitor.properties` file. The default value is `history`.

History Mode

History mode persists information (sending it to storage) about a request after a request has completed. This mode is appropriate in cases where a user is most interested in analyzing request data after the fact and doesn't require real time updates.

Live Mode

Live mode updates request data (sending it to storage) in real time as it changes. This mode is suitable for users who care about what a service is doing now.

Bounding Box

When applicable one of the attributes the monitor extension can capture is the request bounding box. In some cases, such as WMS and WCS requests, capturing the bounding box is easy. However in other cases such as WFS it is not always possible to 100% reliably capture the bounding box. An example being a WFS request with a complex filter element.

How the bounding box is captured is controlled by the `bboxMode` property in the `monitor.properties` file. It can have one of the following values.

- **none** - No bounding box information is captured.
- **full** - Bounding box information is captured and heuristics are applied for WFS requests.
- **no_wfs** - Bounding box information is captured except for WFS requests.

Part of a bounding box is a coordinate reference system (crs).Similar to the WFS case it is not always straight forward to determine what the crs is. For this reason the `bboxCrs` property is used to configure a default crs to be used. The default value for the property is "EPSG:4326" and will be used in cases where all lookup heuristics fail to determine a crs for the bounding box.

Request Body Size

The monitor extension will capture the contents of the request body when a body is specified as is common with a PUT or POST request. However since a request body can be large the extension limits the amount captured to the first 1024 bytes by default.

A value of `0` indicates that no data from the request body should be captured. A value of `-1` indicates that no limit should be placed on the capture and the entire body content should be stored.

This limit is configurable with the `maxBodySize` property of the `monitor.properties` file.

Note: When using database persistence it is important to ensure that the size of the body field in the database can accommodate the `maxBodySize` property.

Request Filters

By default not all requests are monitored. Those requests excluded include any web admin requests or any *Monitor Query API* requests. These exclusions are configured in the `filter.properties` file:

```
/rest/monitor/**
/web/**
```

These default filters can be changed or extended to filter more types of requests. For example to filter out all WFS requests the following entry is added:

```
/wfs
```

How to determine the filter path

The contents of `filter.properties` are a series of ant-style patterns that are applied to the *path* of the request. Consider the following request:

```
http://localhost:8080/geoserver/wms?request=getcapabilities
```

The path of the above request is `/wms`. In the following request:

```
http://localhost:8080/geoserver/rest/workspaces/topp/datastores.xml
```

The path is `/rest/workspaces/topp/datastores.xml`.

In general, the path used in filters is comprised of the portion of the URL after `/geoserver` (including the preceding `/`) and before the query string `?`:

```
http://<host>:<port>/geoserver/<path>?<queryString>
```

Note: For more information about ant-style pattern matching, see the Apache Ant manual.

Samples

monitor.properties

```
# storage and mode
storage=memory
mode=history

# request body capture
maxBodySize=1024

# bounding box capture
bboxMode=no_wfs
bboxCrs=EPSG:4326
```

filter.properties

```
# filter out monitor query api requests
/rest/monitor/**

# filter out all web requests
/web
/web/**

# filter out requests for WCS service
/wcs
```

15.10.5 Audit Logging

The history mode logs all requests into a database. This can put a very significant strain on the database and can lead to insertion issues as the request table begins to host millions of records.

As an alternative to the history mode it's possible to enable the auditing logger, which will log the details of each request in a file, which is periodically rolled. Secondary applications can then process these log files and built ad-hoc summaries off line.

Configuration

The `monitor.properties` file can contain the following items to enable and configure file auditing:

```
audit.enabled=true
audit.path=/path/to/the/logs/directory
audit.roll_limit=20
```

The `audit.enable` is used to turn on the logger (it is off by default). The `audit.path` is the directory where the log files will be created. The `audit.roll_limit` is the number of requests logged into a file before rolling happens. The files are also automatically rolled at the beginning of each day.

In clustered installations with a shared data directory the audit path will need to be different for each node. In this case it's possible to specify the audit path by using a JVM system variable, add the following to the JVM startup options and it will override whatever is specified in `monitor.properties`:

```
-DGEOSERVER_AUDIT_PATH=/path/to/the/logs/directory
```

Log Files

The log directory will contain a number of log files following the `geoserver_audit_yyyymmdd_nn.log` pattern. The `nn` is increased at each roll of the file. The contents of the log directory will look like:

```
geoserver_audit_20110811_2.log
geoserver_audit_20110811_3.log
geoserver_audit_20110811_4.log
geoserver_audit_20110811_5.log
geoserver_audit_20110811_6.log
geoserver_audit_20110811_7.log
geoserver_audit_20110811_8.log
```

By default each log file contents will be a xml document looking like the following:

```
<?xml version="1.0" encoding="UTF-8" ?>
<Requests>
        <Request id="168">
           <Service>WMS</Service>
           <Version>1.1.1</Version>
           <Operation>GetMap</Operation>
           <SubOperation></SubOperation>
           <Resources>GeoSolutions:elba-deparea</Resources>
           <Path>/GeoSolutions/wms</Path>
           <QueryString>LAYERS=GeoSolutions:elba-deparea&amp;STYLES=&amp;FORMAT=image/
↪png&amp;TILED=true&amp;TILESORIGIN=9.916,42.312&amp;SERVICE=WMS&amp;VERSION=1.1.1&
↪amp;REQUEST=GetMap&amp;EXCEPTIONS=application/vnd.ogc.se_inimage&amp;SRS=EPSG:4326&
↪amp;BBOX=9.58375,42.64425,9.916,42.9765&amp;WIDTH=256&amp;HEIGHT=256</QueryString>
           <HttpMethod>GET</HttpMethod>
           <StartTime>2011-08-11T20:19:28.277Z</StartTime>
           <EndTime>2011-08-11T20:19:28.29Z</EndTime>
           <TotalTime>13</TotalTime>
           <RemoteAddr>192.168.1.5</RemoteAddr>
           <RemoteHost>192.168.1.5</RemoteHost>
           <Host>demo1.geo-solutions.it</Host>
           <RemoteUser>admin</RemoteUser>
           <ResponseStatus>200</ResponseStatus>
           <ResponseLength>1670</ResponseLength>
           <ResponseContentType>image/png</ResponseContentType>
           <Failed>false</Failed>
        </Request>
        ...
</Requests>
```

Customizing Log Contents

The log contents are driven by three FreeMarker templates.

`header.ftl` is used once when a new log file is created to form the first few lines of the file. The default header template is:

```
<?xml version="1.0" encoding="UTF-8" ?>
<Requests>
```

`content.ftl` is used to write out the request details. The default template dumps all the known fields about the request:

```
<#escape x as x?xml>
<Request id="${id!""}">
   <Service>${service!""}</Service>
   <Version>${owsVersion!""}</Version>
   <Operation>${operation!""}</Operation>
   <SubOperation>${subOperation!""}</SubOperation>
   <Resources>${resourcesList!""}</Resources>
   <Path>${path!""}</Path>
   <QueryString>${queryString!""}</QueryString>
   <#if bodyAsString??>
   <Body>
   ${bodyAsString}
   </Body>
   </#if>
   <HttpMethod>${httpMethod!""}</HttpMethod>
   <StartTime>${startTime?datetime?iso_utc_ms}</StartTime>
   <EndTime>${endTime?datetime?iso_utc_ms}</EndTime>
   <TotalTime>${totalTime}</TotalTime>
   <RemoteAddr>${remoteAddr!""}</RemoteAddr>
   <RemoteHost>${remoteHost!""}</RemoteHost>
   <Host>${host}</Host>
   <RemoteUser>${remoteUser!""}</RemoteUser>
   <ResponseStatus>${responseStatus!""}</ResponseStatus>
   <ResponseLength>${responseLength?c}</ResponseLength>
   <ResponseContentType>${responseContentType!""}</ResponseContentType>
   <CacheResult>${cacheResult!""}</CacheResult>
   <MissReason>${missReason!""}</MissReason>
   <#if error??>
   <Failed>true</Failed>
   <ErrorMessage>${errorMessage!""}</ErrorMessage>
   <#else>
   <Failed>false</Failed>
   </#if>
</Request>
</#escape>
```

`footer.ftl` is executed just once when the log file is closed to build the last few lines of the file. The default footer template is:

```
</Requests>
```

The administrator is free to provide alternate templates, they can be placed in the same directory as `monitor.properties`, with the same names as above. GeoServer will pick them up automatically.

15.10.6 Monitor Query API

The monitor extension provides a simple HTTP-based API for querying request information. It allows retrieving individual request records or sets of request records, in either HTML or CSV format. Records can be filtered by time range and the result set sorted by any field. Large result sets can be paged over multiple queries.

Examples

The following examples show the syntax for common Monitoring queries.

All requests as HTML

The simplest query is to retrieve an HTML document containing information about all requests:

```
GET http://localhost:8080/geoserver/rest/monitor/requests.html
```

All requests as CSV

Request information can be returned in CSV format, for easier post-processing:

```
GET http://localhost:8080/geoserver/rest/monitor/requests.csv
```

Request bodies containing newlines are handled with quoted text. If your CSV reader doesn't handle quoted newlines, it will not work correctly.

All requests as PKZip

A PKZip archive containing the CSV file above, with all the request bodies and errors as separate files:

```
GET http://localhost:8080/geoserver/rest/monitor/requests.zip
```

All requests as MS Excel

A Microsoft Excel spreadsheet containing the same information as the CSV file:

```
GET http://localhost:8080/geoserver/rest/monitor/requests.xls
```

Requests during a time period

Requests can be filtered by date and time range:

```
GET http://localhost:8080/geoserver/rest/monitor/requests.html?from=2010-06-20&
↪to=2010-07-20
GET http://localhost:8080/geoserver/rest/monitor/requests.html?from=2010-06-
↪20T2:00:00&to=2010-06-20T16:00:00
```

Request set paging

Large result sets can be paged over multiple queries:

```
GET http://localhost:8080/geoserver/rest/monitor/requests.html?count=100
GET http://localhost:8080/geoserver/rest/monitor/requests.html?count=100&offset=100
GET http://localhost:8080/geoserver/rest/monitor/requests.html?count=100&offset=200
GET http://localhost:8080/geoserver/rest/monitor/requests.html?count=100&offset=300
```

Single request

An individual request can be retrieved by specifying its ID:

```
GET http://localhost:8080/geoserver/rest/monitor/requests/12345.html
```

API Reference

There are two kinds of query: one for single requests, and one for sets of requests.

Single Request Query

A query for a single request record has the structure:

```
GET http://<host>:<port>/geoserver/rest/monitor/requests/<id>.<format>
```

where `id` is the numeric identifier of a single request, and `format` specifies the representation of the returned result as one of:

- `html` - an HTML table.
- `csv` - a Comma Separated Values table.
- `zip` - PKZip archive containing CSV as above, plus plain text of errors and request body.
- `xls` - Microsoft Excel spreadsheet.

Note: An alternative to specifying the returned representation with the `format` extension is to use the http `Accept` header and specify the MIME type as one of:

- `text/html`
- `application/csv`
- `application/zip`
- `application/vnd.ms-excel`

See the HTTP specification for more information about the `Accept` header.

Request Set Query

The structure of a query for a set of requests is:

```
GET http://<host>:<port>/geoserver/rest/monitor/requests.<format>[?parameter{&
↪parameter}]
```

where `format` is as described above, and `parameter` is one or more of the parameters listed below.

The request set query accepts various parameters that control what requests are returned and how they are sorted. The available parameters are:

count Parameter

Specifies how many records should be returned.

Syntax	Example
`count=<integer>`	requests.html?count=100

offset Parameter

Specifies where in the result set records should be returned from.

Syntax	Example
`offset=<integer>`	requests.html?count=100&offset=500

live Parameter

Specifies that only live (currently executing) requests be returned.

Syntax	Example
`live=<yes|no|true|false>`	requests.html?live=yes

This parameter relies on a *Monitor Mode* being used that maintains real time request information (either **live** or **mixed**).

from Parameter

Specifies an inclusive lower bound on the timestamp for the start of a request. The timestamp can be specified to any desired precision.

Syntax	Example
`from=<timestamp>`	requests.html?from=2010-07-23T16:16:44
	requests.html?from=2010-07-23

to Parameter

Specifies an inclusive upper bound on the timestamp for the start of a request. The timestamp can be specified to any desired precision.

Syntax	Example
`to=<timestamp>`	requests.html?to=2010-07-24T00:00:00
	requests.html?to=2010-07-24

order Parameter

Specifies which request attribute to sort by, and optionally specifies the sort direction.

Syntax	Example
`order=<attribute>[;<ASC|DESC>]`	requests.html?order=path
	requests.html?order=startTime;DESC
	requests.html?order=totalTime;ASC

15.10.7 GeoIP

The monitor extension has the capability to integrate with the MaxMind GeoIP database in order to provide geolocation information about the origin of a request. This functionality is not enabled by default.

Note: At this time only the freely available GeoLite City database is supported.

Enabling GeoIP Lookup

In order to enable the GeoIP lookup capabilities

1. Download the GeoLite City database.
2. Uncompress the file and copy `GeoLiteCity.dat` to the `monitoring` directory.
3. Restart GeoServer.

15.11 NetCDF

15.11.1 Adding a NetCDF data store

To add a NetCDF data store the user must go to *Stores –> Add New Store –> NetCDF*.

Fig. 15.30: *NetCDF in the list of raster data stores*

15.11.2 Configuring a NetCDF data store

Option	Description
`Workspace`	
`Data Source Name`	
`Description`	
`Enabled`	
`URL`	

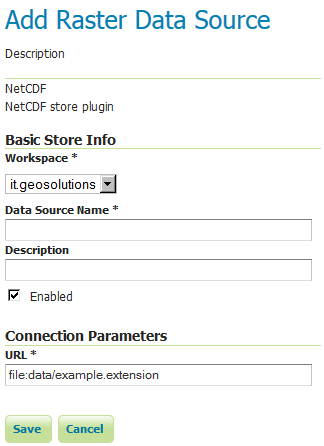


Fig. 15.31: *Configuring a NetCDF data store*

15.11.3 Notes on supported NetCDFs

The NetCDF plugin for GeoServer supports gridded NetCDF files having dimensions following the COARDS convention (custom, Time, Elevation, Lat, Lon). The NetCDF plugin supports plain NetCDF datasets (.nc files) as well .ncml files (which aggregate and/or modify one or more datasets) and Feature Collections. It supports Forecast Model Run Collection Aggregations (FMRC) either through the NCML or Feature Collection syntax. It supports an unlimited amount of custom dimensions, including runtime.

ToolsUI is an useful java tool developed by UCAR which can be useful for a preliminary check on your dataset. Opening a sample NetCDF using that tool will show an output like this in the Viewer tab:

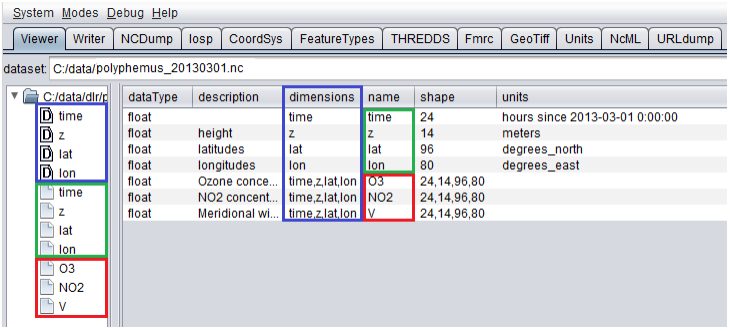


Fig. 15.32: *NetCDF viewer in ToolsUI*

- This dataset has 4 dimensions (time, z, lat, lon, marked by the D icon in the left side of the GUI. They have been marked by a blue rectangle in the screenshot).
- Each dimension has an associated independent coordinate variable (marked by the green rectangle).
- Finally, the dataset has 3 geophysical variables, marked by a red rectangle, each having 4 dimensions.

The NetCDF plugin fully supports datasets where each variable's axis is identified by an independent coordinate variable, as shown in the previous example. There is limited support for coordinate variables with two dimensions (see *Two-Dimensional Coordinate Variables*), either as part of the definition of a plain dataset (such as y,x/rows,cols/i,j/...) or the result of an aggregation (such as time,runtime - in the case of a runtime aggregation).

A similar dataset will look like this in ToolsUI. Look at the red marked latitude and longitude coordinate variables, each one identified by a y,x 2D matrix.

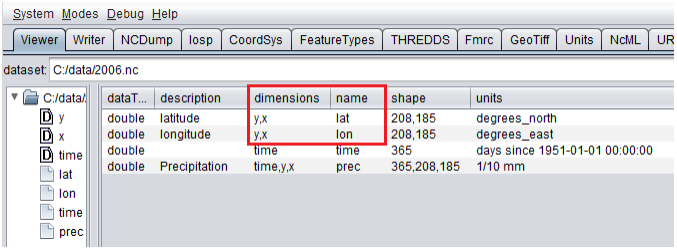


Fig. 15.33: *NetCDF viewer in ToolsUI for 2D coordinate variables*

15.11.4 Two-Dimensional Coordinate Variables

Two-dimension coordinate variables are exposed in GeoServer as single dimensions. Their domain is exposed in GetCapabilities as a flat list of possible values. However, they imply an interdependence between the different dimensions, where some combinations of values exist (have data) and other combinations do not. For example:

Runtime		Time		
		0	1	2
0	1/1/2017	1/1/2017	1/2/2017	1/4/2017
1	1/2/2017	1/2/2017	1/3/2017	XXXX
2	1/3/2017	1/3/2017	XXXX	XXXX

The time dimension would thus be exposed in GeoServer as {1/1/2017, 1/2/2017, 1/3/2017, 1/4/2017}. However, the combinations (runtime=1/1/2017, time=1/3/2017), (runtime=1/2/2017, time=1/1/2017), (runtime=1/2/2017, time=1/4/2017) , (runtime=1/3/2017, time=1/1/2017), (runtime=1/3/2017, time=1/2/2017) and (runtime=1/3/2017, time=1/4/2017) do not exist.

Some additional functionality was introduced to maximally exploit two-dimensional coordinate variables:

- With requests that do not specify certain dimension values, we want to select default values that makes sense with regards to the dimensions values that *were* specified in the request. More specifically we want the maximum or minimum of the domain that matches the specified request's other dimension values; rather than the maximum or minimum of the entire domain.
- The user may want to query which combination of dimension values do exist and which don't. This can be done through an Auxiliary Vector Store that publishes the entire index.

A number of system properties allow us to configure this behavior:

- **`org.geotools.coverage.io.netcdf.param.max`** A comma separated list of dimensions that must be maximised when their value is absent in the request. In the layer configuration, the default value of these dimensions must be set to 'Built-in'.
- **`org.geotools.coverage.io.netcdf.param.min`** A comma separated list of dimensions that must be minimised when their value is absent in the request. In the layer configuration, the default value of these dimensions must be set to 'Built-in'.
- **`org.geotools.coverage.io.netcdf.auxiliary.store`** Set to TRUE to display the 'NetCDF Auxiliary Store' option in Geoserver. A NetCDF Auxiliary Store must be published *after* publishing the actual NetCDF store.

The NetCDF Auxiliary Store returns a WFS record like this for each possible combination of dimension values that do not include the two prime spatial dimensions:

```
<topp:my-aux-store gml:id="1">
 <topp:the_geom>
  <gml:Polygon srsName="http://www.opengis.net/gml/srs/epsg.xml#4326" srsDimension="2
↪">
   <gml:exterior><gml:LinearRing>
   <gml:posList>259.96003054 -0.04 259.96003054 70.04 310.03999998 70.04 310.03999998␣
↪-0.04 259.96003054   -0.04</gml:posList>
   </gml:LinearRing></gml:exterior>
  </gml:Polygon>
 </topp:the_geom>
 <topp:imageindex>160</topp:imageindex>
 <topp:depth>0.0</topp:depth>
 <topp:time>2017-01-01T00:00:00Z</topp:time>
 <topp:runtime>2017-01-02T00:00:00Z</topp:runtime>
</topp:my-aux-store>
```

15.11.5 Supporting Custom NetCDF Coordinate Reference Systems

Grid Mapping attributes

Starting with GeoServer 2.8.x, NetCDF related modules (both NetCDF/GRIB store, imageMosaic store based on NetCDF/GRIB dataset and NetCDF output format) allow to support custom Coordinate Reference Systems and Projections. As reported in the NetCDF CF documentation, Grid mappings section a NetCDF CF file may expose gridmapping attributes to describe the underlying projection. A *grid_mapping* attribute in the variable refers to the name of a variable containing the grid mapping definition.

The GeoTools NetCDF machinery will parse the attributes (if any) contained in the underlying NetCDF dataset to setup an OGC CoordinateReferenceSystem object. Once created, a CRS lookup will be made to identify a custom EPSG (if any) defined by the user to match that Projection. In case the NetCDF gridMapping is basically the same of the one exposed as EPSG entry but the matching doesn't happen, you may consider tuning the comparison tolerance: See *Coordinate Reference System Configuration*, *Increase Comparison Tolerance section*.

User defined NetCDF Coordinate Reference Systems with their custom EPSG need to be provided in `user_projections\netcdf.projections.properties` file inside your data directory (you have to create that file if missing).

A sample entry in that property file could look like this:

> 971835=PROJCS[“albers_conical_equal_area”, GEOGCS[“unknown”, DATUM[“unknown”, SPHEROID[“unknown”, 6378137.0, 298.2572221010042]], PRIMEM[“Greenwich”, 0.0], UNIT[“degree”, 0.017453292519943295], AXIS[“Geodetic longitude”, EAST], AXIS[“Geodetic latitude”, NORTH]], PROJECTION[“Albers_Conic_Equal_Area”], PARAMETER[“central_meridian”, -126.0], PARAMETER[“latitude_of_origin”, 45.0], PARAMETER[“standard_parallel_1”, 50.0], PARAMETER[“false_easting”, 1000000.0], PARAMETER[“false_northing”, 0.0], PARAMETER[“standard_parallel_2”, 58.5], UNIT[“m”, 1.0], AXIS[“Easting”, EAST], AXIS[“Northing”, NORTH], AUTHORITY[“EPSG”,”971835”]]

Note: Note the “unknown” names for GEOGCS, DATUM and SPHEROID elements. This is how the underlying NetCDF machinery will name custom elements.

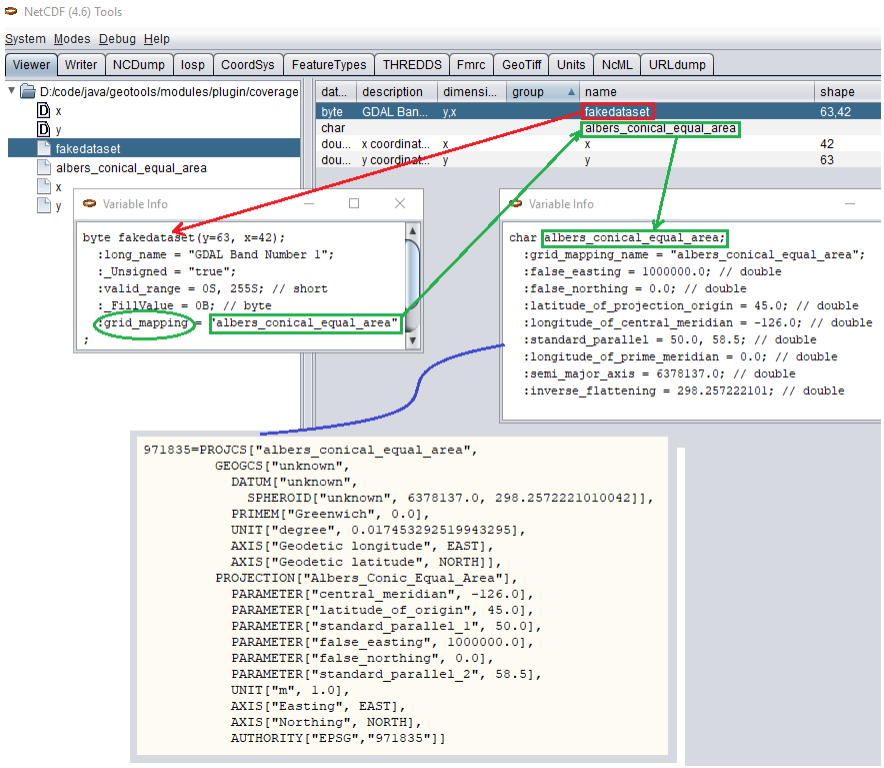


Fig. 15.34: *Grid Mapping and related custom EPSG definition*

Note: Note the number that precedes the WKT. This will determine the EPSG code. So in this example, the EPSG code is 971835.

Note: When dealing with records indexing based on PostGIS, make sure the custom code isn't greater than 998999. (It tooks us a while to understand why we had some issues with custom codes using PostGIS as granules index. Some more details, here)

Note: If a parameter like "central_meridian" or "longitude_of_origin" or other longitude related value is outside the range [-180,180], make sure you adjust this value to belong to the standard range. As an instance a Central Meridian of 265 should be set as -95.

You may specify further custom NetCDF EPSG references by adding more lines to that file.

1. Insert the code WKT for the projection at the end of the file (on a single line or with backslash characters):

```
971835=PROJCS["albers_conical_equal_area", \
      GEOGCS["unknown", \
            DATUM["unknown", \
              SPHEROID["unknown", 6378137.0, 298.2572221010042]],  \
        PRIMEM["Greenwich", 0.0], \
            UNIT["degree", 0.017453292519943295], \
            AXIS["Geodetic longitude", EAST], \
```

```
    AXIS["Geodetic latitude", NORTH]], \
  PROJECTION["Albers_Conic_Equal_Area"], \
  PARAMETER["central_meridian", -126.0], \
  PARAMETER["latitude_of_origin", 45.0], \
  PARAMETER["standard_parallel_1", 50.0], \
  PARAMETER["false_easting", 1000000.0], \
  PARAMETER["false_northing", 0.0], \
  PARAMETER["standard_parallel_2", 58.5], \
  UNIT["m", 1.0], \
  AXIS["Easting", EAST], \
  AXIS["Northing", NORTH], \
  AUTHORITY["EPSG","971835"]]
```

2. Save the file.
3. Restart GeoServer.
4. Verify that the CRS has been properly parsed by navigating to the *SRS List* page in the *Web administration interface*.
5. If the projection wasn't listed, examine the logs for any errors.

Specify an external file through system properties

You may also specify the NetCDF projections definition file by setting a **Java system property** which links to the specified file. As an instance: `-Dnetcdf.projections.file=/full/path/of/the/customfile.properties`

WKT Attributes

Some NetCDFs may include a text attribute containing the WKT definition of a Coordinate Reference System. When present, it will be parsed by GeoServer to setup a CRS and a lookup will be performed to see if any EPSG is matching it.

- **spatial_ref** GDAL *spatial_ref* attribute
- **esri_pe_string** An attribute being defined by NetCDF CERP Metadata Convention

15.11.6 NetCDF files in read-only directories

GeoServer creates hidden index files when accessing NetCDF files. Because these index files are created in the same directory as each NetCDF file, GeoServer will fail to publish NetCDF files if it lacks write access the containing directory.

To permit access to NetCDF files in read-only directories, specify an alternate writeable directory for NetCDF index files by setting the `NETCDF_DATA_DIR` Java system property:

```
-DNETCDF_DATA_DIR=/path/to/writeable/index/file/directory
```

15.11.7 Supporting Custom NetCDF Units

The NetCDF format expresses units using a syntax that is not always understood by our unit parser, and often, uses unit names using unrecognized symbols or that simply unknown to it. The system already

comes with some smarts, but in case a unit is not recognized, it's possible to act on the configuration and extend it.

There are two property files that can be setup in order to modify unit magement, one is an alias file, the other is a replacement file:

- An "alias" is a different symbol/name for a base unit (e.g., instead of using "g" the NetCDF files might be using "grammes")
- A (text) "replacement" is used when the unit is a derived one, needing a full expression, or the syntax of the unit is simply unrecognized

The alias file is called `netcdf-unit-aliases.properties`, if not provided these contents are assumed:

```
# Aliases for unit names that can in turn be used to build more complex units
Meter=m
meter=m
Metre=m
microgram=µg
micrograms=µg
nanograms=ng
degree=deg
percentage=%
celsius=°C
````
```

The replacement file is called `netcdf-unit-replacements.properties`, if not provided the following contents are assumed:

```
micrograms\ per\ cubic\ meter=µg*m^-3
DU=µmol*m^-2*446.2
m2=m^2
m3=m^3
s2=s^2
```

Both files express the NetCDF unit as the key, and the standard symbol or replacement text as the value.

It is possible to place the files in three different locations:

- If the `NETCDF_UNIT_ALIASES` and/or `NETCDF_UNIT_REPLACEMENTS` system variables are defined, the respective files will be looked up at the specified location (must be full paths, including the file name)
- If the above are missing and external NetCDF data dir is defined via `NETCDF_DATA_DIR` then the files will be looked up in there
- If the above are missing the root of the GeoServer data directory will be searched
- If none of the above provide a file, then the built-in configuration will be used

15.11.8 Migrating mosaics with H2 NetCDF index files to a centralized index

By default the NetCDF reader creates a hidden directory, either as a sidecar or in the NetCDF data dir, containing a low level index file to speed up slices lookups, as well as a H2 database containing information about slice indexes and dimensions associated to them. This H2 store is opened and closed every time the associated NetCDF is read, causing less than optimal performance in map rendering.

As an alternative, it's possible to store all slice metadata from H2 to a centralized database, and have GeoServer manage the store connecting to it, thus keeping it always open. Some work is in order to make that happen thought.
```

First, the image mosaic needs a `indexer.xml` and a NetCDF auxiliary file, describing the coverages structure. These two files can be generated using a tool available in the GeoServer classpath, that one has to invoke from the command line.

Given a sample NetCDF file, you can get into the mosaic directory and run the **CreateIndexer** tool (for the NetCDF projection files, see above):

```
java -cp <path-to-geoserver>/WEB-INF/lib/*.jar org.geotools.coverage.io.netcdf.tools.
↪CreateIndexer <path-to-sample-nc-file> -p <path-to-netcdf-projections> [<path-to-
↪output-directory>]
```

This will generate the files and it's going to be good enough if each NetCDF contains the same coverages. If instead there are different NetCDF files containing different coverages in the same mosaic, you'll have to:

- run the above command using a different sample NetCDF file for each coverage, generating the output in different folders,
- manually merge them into a unified `indexer.xml` and `_auxiliary.xml` that will be placed in the mosaic directory.

Once those files are in position, a second tool can be run in order to migrate all H2 files to a centralized datatabase. First, prepare a property file with connection parameters to the target index database. For example, it could be a `netcdf_index.properties` file with the following contents:

```
SPI=org.geotools.data.postgis.PostgisNGDataStoreFactory
host=localhost
port=5432
database=netcdfidx
schema=public
user=user
passwd=pwd
Loose\ bbox=true
Estimated\ extends=false
validate\ connections=true
Connection\ timeout=10
preparedStatements=true
max\ connections=20
```

Then, in order to migrate a specific mosaic, run the **H2Migrate** tool:

```
java -cp <path-to-geoserver>/WEB-INF/lib/*.jar org.geotools.coverage.io.netcdf.tools.
↪H2Migrate -m <path-to-mosaic-directory> -is <indexPropertyFile> -isn
↪<storeNameForIndex> -v
```

This will connect to the target store using the information in indexPropertyFile, locate the granules to be migrated inspecting the mosaic contents, create a `netcdf_index.properties` file with `StoreName=storeNameForIndex` and update the mosaic to use it (basically, update the indexer.xml and all coverage property files to have a `AuxiliaryDatastoreFile` property pointing to `netcdf_indexer.properties`, as well ensure that there is a `AuxiliaryFile` property pointing to `_auxiliary.xml`).

It will also generate two files, `migrated.txt` and `h2.txt`:

- `migrated.txt` contains the list of files successfully migrated, for audit purposes
- `h2.txt` the list of H2 database files that can now be removed. The tool won't do it automatically to ensure that the migration, but with this one one could automate removal, e.g., on Linux a simple `cat h2.txt | rm` would do the trick (the `<name>.log.db` files change name often, it's likely that they will have to be hunted down and removed with other means, e.g. if on Linux, using the "find").

If the mosaic to be migrated is backed by a **OpenSearch** index, then the tool won't be able to open the mosaic (it would require running inside GeoServer), so the connection parameters will have to be provided in a second property file, along with the list of tables containing the granules paths in the "location" attribute, e.g.:

> java -cp <path-to-geoserver>/WEB-INF/lib/*.jar org.geotools.coverage.io.netcdf.tools.H2Migrate -m <path-to-mosaic-directory> -ms <mosaicStorePropertyFile> -mit granule -is <indexPropertyFile> -isn <storeNameForIndex> -v

The tool supports other options as well, they can be discovered by running the tool without any parameter:

```
java -cp <path-to-geoserver>/WEB-INF/lib/*.jar org.geotools.coverage.io.netcdf.tools.
↪H2Migrate -m <path-to-mosaic-directory> -is <indexPropertyFile> -isn
↪<storeNameForIndex> -v
```

15.12 NetCDF Output Format

This plugin adds the ability to encode WCS 2.0.1 multidimensional output as a NetCDF file using the Unidata NetCDF Java library.

15.12.1 Getting a NetCDF output file

Request NetCDF output by specifying `format=application/x-netcdf` in a `GetCoverage` request:

```
http://localhost:8080/geoserver/wcs?service=WCS&version=2.0.1&request=GetCoverage&
↪coverageid=it.geosolutions__V&format=application/x-netcdf...
```

15.12.2 Current limitations

- Input coverages/slices should share the same bounding box (lon/lat coordinates are the same for the whole ND cube).
- NetCDF output will be produced only when input coverages come from a StructuredGridCoverage2D reader (this allows to query the GranuleSource to get the list of granules in order to setup dimensions slices for each sub-coverage).

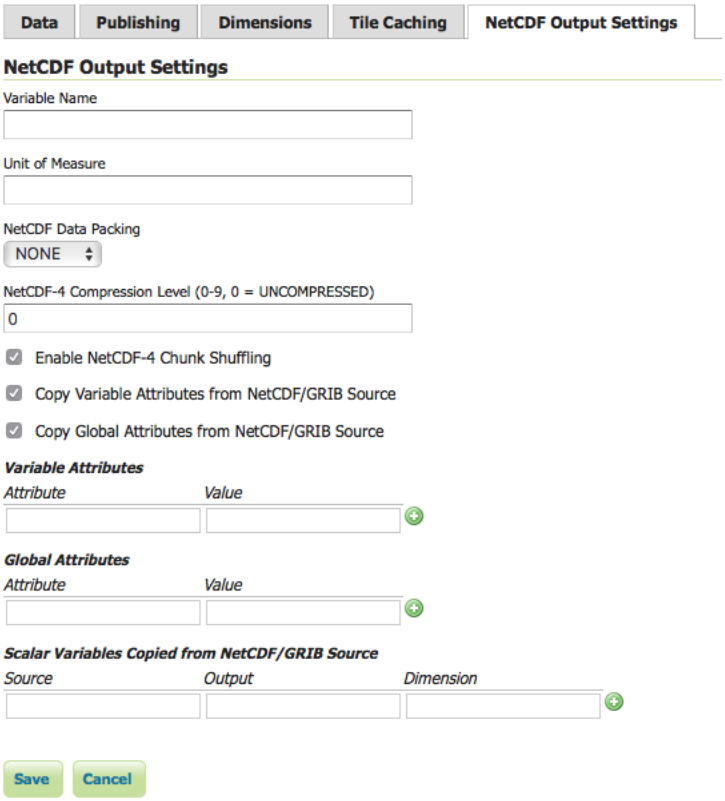
15.12.3 NetCDF-4

NetCDF-4 output is supported but requires native libraries (see Installing required NetCDF-4 Native libraries). NetCDF-4 adds support for compression. Use `format=application/x-netcdf4` to request NetCDF-4 output.

15.12.4 Settings

NetCDF output settings can be configured for each raster layer. The similar section in the *Global Settings* page configures the default settings used for newly created raster layers.

- **Variable Name (optional)**
  - Sets the NetCDF variable name.
  - Does not change the layer name, which can be configured in the Data tab.



- **Variable Unit of Measure (optional)**
  - Sets the NetCDF `uom` attribute.
- **Data Packing**
  - Lossy compression by storing data in reduced precision.
  - One of *NONE*, *BYTE*, *SHORT*, or *INT*.
- **NetCDF-4 Compression Level**
  - Lossless compression.
  - Level is an integer from 0 (no compression, fastest) to 9 (most compression, slowest).
- **NetCDF-4 Chunk Shuffling**
  - Lossless byte reordering to improve compression.
- **Copy Variable Attributes from NetCDF/GRIB Source**
  - Most attributes are copied from the source NetCDF/GRIB variable.
  - Some attributes such as `coordinates` and `missing_value` are skipped as these may no longer be valid.
  - For an ImageMosaic, one granule is chosen as the source.
- **Copy Global Attributes from NetCDF/GRIB Source**
  - Attributes are copied from the source NetCDF/GRIB global attributes.
  - For an ImageMosaic, one granule is chosen as the source.

- **Variable Attributes**
  - Values are encoded as integers or doubles if possible, otherwise strings.
  - Values set here overwrite attributes set elsewhere, such as those copied from a source NetCDF/GRIB variable.
- **Global Attributes**
  - Values are encoded as integers or doubles if possible, otherwise strings.
- **Scalar Variables Copied from NetCDF/GRIB Source**
  - Source specifies the name of the source variable in a NetCDF file or the `toolsUI` view of a GRIB file; only scalar source variables are supported.
  - Output specifies the name of the variable in the output NetCDF file.
  - If only one of Source or Output is given, the other is taken as the same.
  - Dimension is either blank to simply copy the source scalar from one granule, or the name of one output NetCDF dimension to cause values to be copied from multiple granules (such as those from an ImageMosaic over a non-spatial dimension) into a one-dimensional variable. The example above copies a single value from multiple `reftime` scalars into `forecast_reference_time` dimensioned by `time` in an ImageMosaic over time.
  - For an ImageMosaic, one granule is chosen as the source for variable attributes.

15.12.5 CF Standard names support

Note that the output name can also be chosen from the list of CF Standard names. Check CF standard names page for more info on it.

Once you click on the dropdown, you may choose from the set of available standard names.

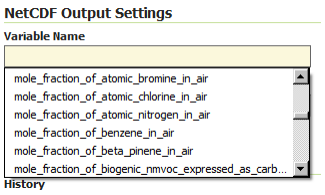


Fig. 15.35: *NetCDF CF Standard names list*

Note that once you specify the standard name, the unit will be automatically configured, using the canonical unit associated with that standard name.

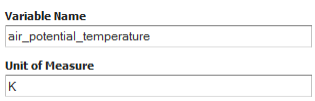


Fig. 15.36: *NetCDF CF Standard names and canonical unit*

The list of standard names is populated by taking the entries from a standard name table xml. At time of writing, a valid example is available Here

You have three ways to provide it to GeoServer.

1. Add a `-DNETCDF_STANDARD_TABLE=/path/to/the/table/tablename.xml` property to the startup script.
2. Put that xml file within the `NETCDF_DATA_DIR` which is the folder where all NetCDF auxiliary files are located. (More info)
3. Put that xml file within the `GEOSERVER_DATA_DIR`.

Note: Note that for the 2nd and 3rd case, file name must be **cf-standard-name-table.xml**.

15.13 OGR based WFS Output Format

The ogr2ogr based output format leverages the availability of the ogr2ogr command to allow the generation of more output formats than GeoServer can natively produce. The basics idea is to dump to the file system a file that ogr2ogr can translate, invoke it, zip and return the output of the translation.

15.13.1 Out of the box behaviour

Out of the box the plugin assumes the following:

- ogr2ogr is available in the path
- the GDAL_DATA variable is pointing to the GDAL data directory (which stores the spatial reference information for GDAL)

In the default configuration the following formats are supported:

- MapInfo in TAB format
- MapInfo in MIF format
- Un-styled KML
- CSV (without geometry data dumps)

The list might be shorter if ogr2ogr has not been built with support for the above formats.

Once installed in GeoServer four new GetFeature output formats will be available, in particular, `OGR-TAB`, `OGR-MIF`, `OGR-KML`, `OGR-CSV`.

15.13.2 ogr2ogr conversion abilities

The ogr2ogr utility is usually able to convert more formats than the default setup of this output format allows for, but the exact list depends on how the utility was built from sources. To get a full list of the formats available by your ogr2ogr build just run:

```
ogr2ogr --help
```

and you'll get the full set of options usable by the program, along with the supported formats. For example, the above produces the following output using the FWTools 2.2.8 distribution (which includes ogr2ogr among other useful information and conversion tools):

```
Usage: ogr2ogr [--help-general] [-skipfailures] [-append] [-update] [-gt n]
            [-select field_list] [-where restricted_where]
            [-sql <sql statement>]
            [-spat xmin ymin xmax ymax] [-preserve_fid] [-fid FID]
            [-a_srs srs_def] [-t_srs srs_def] [-s_srs srs_def]
            [-f format_name] [-overwrite] [[-dsco NAME=VALUE] ...]
            [-segmentize max_dist]
            dst_datasource_name src_datasource_name
            [-lco NAME=VALUE] [-nln name] [-nlt type] [layer [layer ...]]

-f format_name: output file format name, possible values are:
  -f "ESRI Shapefile"
  -f "MapInfo File"
  -f "TIGER"
  -f "S57"
  -f "DGN"
  -f "Memory"
  -f "BNA"
  -f "CSV"
  -f "GML"
  -f "GPX"
  -f "KML"
  -f "GeoJSON"
  -f "Interlis 1"
  -f "Interlis 2"
  -f "GMT"
  -f "SQLite"
  -f "ODBC"
  -f "PostgreSQL"
  -f "MySQL"
  -f "Geoconcept"
-append: Append to existing layer instead of creating new if it exists
-overwrite: delete the output layer and recreate it empty
-update: Open existing output datasource in update mode
-select field_list: Comma-delimited list of fields from input layer to
                    copy to the new layer (defaults to all)
-where restricted_where: Attribute query (like SQL WHERE)
-sql statement: Execute given SQL statement and save result.
-skipfailures: skip features or layers that fail to convert
-gt n: group n features per transaction (default 200)
-spat xmin ymin xmax ymax: spatial query extents
-segmentize max_dist: maximum distance between 2 nodes.
                      Used to create intermediate points
-dsco NAME=VALUE: Dataset creation option (format specific)
-lco  NAME=VALUE: Layer creation option (format specific)
-nln name: Assign an alternate name to the new layer
-nlt type: Force a geometry type for new layer.  One of NONE, GEOMETRY,
     POINT, LINESTRING, POLYGON, GEOMETRYCOLLECTION, MULTIPOINT,
     MULTIPOLYGON, or MULTILINESTRING.  Add "25D" for 3D layers.
     Default is type of source layer.
-a_srs srs_def: Assign an output SRS
-t_srs srs_def: Reproject/transform to this SRS on output
-s_srs srs_def: Override source SRS

Srs_def can be a full WKT definition (hard to escape properly),
or a well known definition (ie. EPSG:4326) or a file with a WKT
definition.
```

The full list of formats that ogr2ogr is able to support is available on the OGR site. Mind that this output format can handle only outputs that are file based and that do support creation. So, for example, you won't be able to use the Postgres output (since it's database based) or the ArcInfo binary coverage (creation not supported).

15.13.3 Customisation

If ogr2ogr is not available in the default path, the GDAL_DATA is not set, or if the output formats needs tweaking, a `ogr2ogr.xml` file can be put in the root of the GeoServer data directory to customize the output format.

The default GeoServer configuration is equivalent to the following xml file:

```
<OgrConfiguration>
  <ogr2ogrLocation>ogr2ogr</ogr2ogrLocation>
  <!-- <gdalData>...</gdalData> -->
  <formats>
    <Format>
      <ogrFormat>MapInfo File</ogrFormat>
      <formatName>OGR-TAB</formatName>
      <fileExtension>.tab</fileExtension>
    </Format>
    <Format>
      <ogrFormat>MapInfo File</ogrFormat>
      <formatName>OGR-MIF</formatName>
      <fileExtension>.mif</fileExtension>
      <option>-dsco</option>
      <option>FORMAT=MIF</option>
    </Format>
    <Format>
      <ogrFormat>CSV</ogrFormat>
      <formatName>OGR-CSV</formatName>
      <fileExtension>.csv</fileExtension>
      <singleFile>true</singleFile>
      <mimeType>text/csv</mimeType>
    </Format>
    <Format>
      <ogrFormat>KML</ogrFormat>
      <formatName>OGR-KML</formatName>
      <fileExtension>.kml</fileExtension>
      <singleFile>true</singleFile>
      <mimeType>application/vnd.google-earth.kml</mimeType>
    </Format>
  </formats>
</OgrConfiguration>
```

The file showcases all possible usage of the configuration elements:

- `ogr2ogrLocation` can be just ogr2ogr if the command is in the path, otherwise it should be the full path to the executable. For example, on a Windows box with FWTools installed it might be:

  ```
  <ogr2ogrLocation>c:\Programmi\FWTools2.2.8\bin\ogr2ogr.exe</ogr2ogrLocation>
  ```

- `gdalData` must point to the GDAL data directory. For example, on a Windows box with FWTools installed it might be:

```
<gdalData>c:\Programmi\FWTools2.2.8\data</gdalData>
```

- `Format` defines a single format, which is defined by the following tags:
  - `ogrFormat`: the name of the format to be passed to ogr2ogr with the -f option (it's case sensitive).
  - `formatName`: is the name of the output format as advertised by GeoServer
  - `fileExtension`: is the extension of the file generated after the translation, if any (can be omitted)
  - `option`: can be used to add one or more options to the ogr2ogr command line. As you can see by the MIF example, each item must be contained in its own tag. You can get a full list of options by running ogr2ogr –help or by visiting the ogr2ogr web page. Also consider that each format supports specific creation options, listed in the description page for each format (for example, here is the MapInfo one).
  - `singleFile` (since 2.0.3): if true the output of the conversion is supposed to be a single file that can be streamed directly back without the need to wrap it into a zip file
  - `mimeType` (since 2.0.3): the mime type of the file returned when using `singleFile`. If not specified `application/octet-stream` will be used as a default.

15.14 OGR based WPS Output Format

The OGR based WPS output format provides the ability to turn feature collection (vector layer) output types into formats supported by OGR, using the same configuration and same machinery provided by the OGR WFS output format (which should also be installed for the WPS portion to work).

Unlike the WFS case the WPS output formats are receiving different treatment in WPS responses depending on whether they are binary, text, or xml, when the Execute response style chosen by the client is "document":

- Binary types need to be base64 encoded for XML embedding
- Text types need to be included inside a CDATA section
- XML types can be integrated in the response as-is

In order to understand the nature of the output format a new optional configuration element, `<type>`, can be added to the `ogr2ogr.xml` configuration file in order to specify the output nature. The possible values are `binary`, `text`, `xml`, in case the value is missing, `binary` is assumed. Here is an example showing all possible combinations:

```
<OgrConfiguration>
    <ogr2ogrLocation>ogr2ogr</ogr2ogrLocation>
    <!-- <gdalData>...</gdalData> -->
    <formats>
        <Format>
            <ogrFormat>MapInfo File</ogrFormat>
            <formatName>OGR-TAB</formatName>
            <fileExtension>.tab</fileExtension>
            <type>binary</type> <!-- not really required, it's the default -->
        </Format>
        <Format>
            <ogrFormat>MapInfo File</ogrFormat>
            <formatName>OGR-MIF</formatName>
```

```
            <fileExtension>.mif</fileExtension>
            <option>-dsco</option>
            <option>FORMAT=MIF</option>
        </Format>
        <Format>
            <ogrFormat>CSV</ogrFormat>
            <formatName>OGR-CSV</formatName>
            <fileExtension>.csv</fileExtension>
            <singleFile>true</singleFile>
            <mimeType>text/csv</mimeType>
            <option>-lco</option>
            <option>GEOMETRY=AS_WKT</option>
            <type>text</type>
        </Format>
        <Format>
            <ogrFormat>KML</ogrFormat>
            <formatName>OGR-KML</formatName>
            <fileExtension>.kml</fileExtension>
            <singleFile>true</singleFile>
            <mimeType>application/vnd.google-earth.kml</mimeType>
            <type>xml</type>
        </Format>
    </formats>
</OgrConfiguration>
```

15.15 GeoServer Printing Module

The `printing` module for GeoServer allows easy hosting of the Mapfish printing service within a GeoServer instance. The Mapfish printing module provides an HTTP API for printing that is useful within JavaScript mapping applications. User interface components for interacting with the print service are available from the Mapfish and GeoExt projects.

15.15.1 Installation

- Download the extension (named like `geoserver-<version>-printing-plugin.zip`) from the geoserver site download page.
- Extract the contents of the ZIP archive into the `/WEB-INF/lib/` in the GeoServer webapp. For example, if you have installed the GeoServer binary to `/opt/geoserver-2.6/`, the printing extension JAR files should be placed in `/opt/geoserver-2.6/webapps/geoserver/WEB-INF/lib/`.
- After extracting the extension, restart GeoServer in order for the changes to take effect. All further configuration can be done with GeoServer running.

15.15.2 Verifying Installation

On the first startup after installation, GeoServer should create a print module configuration file in *GEOSERVER_DATA_DIR*`/printing/config.yaml`. Checking for this file's existence is a quick way to verify the module is installed properly. It is safe to edit this file; in fact there is currently no way to modify the print module settings other than by opening this configuration file in a text editor.

If the module is installed and configured properly, then you will also be able to retrieve a list of configured printing parameters from http://localhost:8080/geoserver/pdf/info.json . This service must be working properly for JavaScript clients to use the printing service.

Finally, you can test printing in this `sample page`. You can load it directly to attempt to produce a map from a GeoServer running at http://localhost:8080/geoserver/. If you are running at a different host and port, you can download the file and modify it with your HTML editor of choice to use the proper URL.

Warning: This sample script points at the development version of GeoExt. You can modify it for production use, but if you are going to do so you should also host your own, minified build of GeoExt and OpenLayers. The libraries used in the sample are subject to change without notice, so pages using them may change behavior without warning.

15.15.3 MapFish documentation

Configuration

The server side uses a YAML configuration file that defines the page layouts and allowed values. This file is usually called config.yaml.

Here is the general structure:

```
dpis:
  - 254
  - 190
  {...}

?maxSvgWidth: 2048 # set the maximum dimensions to 2048 points, this is useful when␣
↪using MapServer and a maximum dimension is there
?maxSvgHeight: 2048
?integerSvg: false # the library in MapServer <= 5.6 does not support floating point␣
↪values in the SVG coordinate space, set this to true if using a WMS that does not␣
↪support floating point values in SVG coordinates

?ignoreCapabilities: false # assume client is correct and do not load capabilities. ␣
↪This is not recommended to be used unless you it fails when false (false is default)
?maxPrintTimeBeforeWarningInSeconds: 30 # if print jobs take longer than this then a␣
↪warning in the logs will be written along with the spec.
?printTimeoutMinutes: 5 # The maximum time to allow a print job to take before␣
↪cancelling the print job.  The default is 5 (minutes)
?formats:
  - pdf
  - png
  {...}

scales:
  - 25000
  - 50000
  {...}

hosts:
  - {HOST_WHITELIST_DEFINITION}
  {...}

?localHostForward: # For request on map.example.com we build an http request on␣
↪localhost with the header Host=map.example.com, this is to don't pass throw the␣
↪proxy.
```

```
?  from:
?    - map.example.com
?  https2http: True # For above hosts on request on https we build a request on http

?headers: ['Cookie', 'Referer'] # The header that will be copyed to the tiles http␣
↪requests

?keys:
?  - !key
?    host: !dnsMatch
?      host: maps.google.com
?      port: 80
?    domain: !dnsMatch
?      host: localhost
?    key: 1234456
?    id: gmd-xyz


?fonts:
? - {PATH}

?globalParallelFetches: 5
?perHostParallelFetches: 5
?tilecacheMerging: false
?connectionTimeout: 30000           MF_V1.2
?socketTimeout: 180000              MF_V1.2
?outputFilename: Mapfish-print      MF_V1.2
?disableScaleLocking: false
?brokenUrlPlaceholder: default      MF_V2.0
?proxyBaseUrl: http://mapfishprint.org  MF_V2.0
?tmsDefaultOriginX: 0.0f    MF_V2.0
?tmsDefaultOriginY: 0.0f    MF_V2.0

?security:
?  - !basicAuth
?      matcher: !dnsMatch
?        host: www.camptocamp.com
?        post: 443
?      username: xyz
?      password: zyx
?      preemptive: true
?  - !basicAuth
?    username: abc
?    password: bca

layouts:
   {LAYOUT_NAME}:
?   : Mapfish-print.pdf  MF_V1.2
?   metaData:
?     {METADATA_DEFINITION}
?   titlePage:
?     {PAGE_DEFINITION}
    mainPage:
?     rotation: false
      {PAGE_DEFINITION}
?   lastPage:
?     {PAGE_DEFINITION}
  {...}
```

Optional parts are shown with a question mark in the left margin. The question marks must not be put in the configuration file. Their default values is shown.

Note: Sets of values like DPI can be entered in one of two forms:

```
dpi: [1,2,3,...]
```

or

```
dpis:
  - 254
  - 190
```

A chosen DPI value from the above configuration is used in WMS GetMap requests as an added format_options (GeoServer) or map_resolution (MapServer) parameter. This is used for symbol/label-rescaling suitable for high resolution printouts, see *GeoServer format_options specification* (GeoServer 2.1) and MapServer defresolution keyword (MapServer 5.6) for more information.

In general, PDF dimensions and positions are specified in points. 72 points == 1 inch == 25.4 mm.

The list of `{HOST_WHITELIST_DEFINITION}` defines the allowed URLs for getting maps. Its format will be defined in the next sub-section.

The formats element lists the values formats that the server permits. If omitted only 'pdf' is permitted. If the single element '*' (quotes are required) is present then all formats that the server can produce can be requested. The formats the server can produce depends to a large degree on how the Java is configured. PDF is supported on all systems but for image output formats JAI and ImageIO is used which means both must be on the server for them to be available. You can get the list of supported formats by running the standalone client with the –clientConfig flag enabled (you will need to supply a yaml config file as well). If you are using the servlet then do a get info request to see the list of formats (with the '*' as the outputFormats parameter in the config file).

You can have as many layouts as you want. Their name must be unique and will be used on the client side. A layout can have a `titlePage` that will be added at the beginning of the generated document. It cannot contain any map. Same for the `lastPage`, but for the end of the document. The `mainPage` section is mandatory and will be used once for each page requested. The details of a `{PAGE_DEFINITION}` section can be found in another sub-section of this document.

If you want to let the user rotate the map (for a given layout), you have to set the `rotate` field to `true` in the corresponding `mainPage` section.

`globalParallelFetches` and `perHostParallelFetches` are used to tune the parallel loading of the map tiles/images. If you want to disable the parallel loading, set `globalParallelFetches` to 1.

New versions of tilecache added the support for merging multiple layers in a single WMS request. If you want to use this functionality, set the `tilecacheMerging` attribute to true.

`connectionTimeout` and `socketTimeout` (only since MapFish v1.2) can be used to tune the timeouts for reading tiles from map servers.

If the `outputFilename` parameter is defined in the main body then that name will be used by the MapPrintServlet when sending the pdf to the client. It will be the name of the file that the client downloads. If the 'outputFilename' parameter is defined in a layout then that value will override the default name. In both cases the .pdf is optional; if not present the server will append .pdf to the name. In all cases the json request can override the filename defined in the configuration file by posting a 'outputFilename' attribute in the posted JSON. If the outputFilename has ${date}, ${time} or ${dateTime} in it, it will be replaced with the current date using the related DateFormat.get*Instance().format() method. If a pattern is provided it will be passed to SimpleDataFormat for processing. A few examples follow:

- outputFilename: `host-${yyyyMMdd}.pdf` # results in host-20111213.pdf

- outputFilename: `host-${date}` # results in host-Dec_13_2011.pdf (actual output depends on locale of server)
- outputFilename: `host-${dateTime}` # results in host-Dec_13_2011_1:10:50_PM.pdf (actual output depends on locale of server)
- outputFilename: `host-${time}.pdf` # results in host-1:11:14_PM.pdf (actual output depends on locale of server)
- outputFilename: `host-${yyMMdd-hhmmss}` # results in host-111213-011154.pdf (actual output depends on locale of server)

`disableScaleLocking` allows you to bypass the choosing of scale from the available factors, and simply use the suggested value produced inside MapBlock.java.

`brokenUrlPlaceholder` the placeholder image to use in the case of a broken url. By default, when a url request fails, an error is thrown and the pdf process terminates. However if this parameter is set then instead a placeholder image is returned. Non-null values are:

- `default` - use the system default image.
- `throw` - throw an exception.
- <url> - obtain the image from the supplied url. If this url is broken then an exception will be thrown. This can be anytype of valid url from a file url to https url.

`proxyBaseUrl` the optional url of the proxy between mapfish-print and the internet. This is the url base that will be in the info.json response. On occasion the url or port of the web server containing mapfish-print is not the server that is public to the internet and the requests are proxied to the mapfish-print webserver. In this case it is important for the info.json request to return the public URL instead of the url of the webserver.

`tmsDefaultOriginX` By default this is null. If non-null then TmsMapReader will use this as the origin x value if null then the origin will be derived from the maxExtent parameter.

`tmsDefaultOriginY` By default this is null. If non-null then TmsMapReader will use this as the origin y value if null then the origin will be derived from the maxExtent parameter.

Security

Both Keys and Security are options for accessing protected services. Keys are currently for Google maps premium accounts and Security is for other types and is more general Currently only BasicAuth is supported but other strategies can easily be added

```
security:
    - !basicAuth
        matcher: !dnsMatch
          host: www.camptocamp.com
          post: 443
        username: xyz
        password: zyx
        preemptive: true
    - !basicAuth
      username: abc
      password: cba
```

The above example has 2 security configuration. Each option is tested (in order) to see if it can be used for the given URI and if it applies it is used to configure requests for the URI. In the above example the first configuration will be used if the URI matches the hostmatcher provided if not then the second configuration will be applied. The last configuration has no host matcher so it is applied to all URIs.

A basicAuth security configuration consists of 4 options

- matcher - a host matcher for determining which requests need the security to be applied
- username - username for basicauth
- password - password for basicauth
- preemptive - optional, but for cases where the credentials need to be sent without the challenge

Keys

Google maps currently requires a private key to be used (we only support users Google maps premium accounts).

The keys section allows a key to be mapped to hosts. The hosts are identified with host matchers that are described in the <configuration.html#host-whitelist-definition> sub-section.

In addition a domain hostmatcher can be used to select a key based on the domain of the local server. This can be useful if the same configuration is used in a test environment and a production environment with differing domains. For example mapfish.org and mapfish.net.

Finally google maps (for example) requires a client id as well that is associated with the private key. There for in the case of google premium services a legal key would be:

```
keys:
  - !key
    key: yxcvyxvcyxvyx
    id: gme-xxxcs
```

Thanks to the hosts and domain matcher it is possible to have a key for google maps and (for future proofing) a different key for a different service.

Fonts definition

The `fonts` section is optional. It contains the path of the fonts you want to use. The entries can point to files (TTF, OTF, TTC, AFM, PFM) or directories. Don't point to directories containing too many files since it will slow down the start time. By default, PDF gives you access to the following fonts (Cp1252 encoding only):

- Courrier (-Bold, -Oblique, -BoldOblique)
- Helvetica (-Bold, -Oblique, -BoldOblique)
- Times (-Roman, -Bold, -Oblique, -BoldOblique)
- Symbol
- ZapfDingbats

Host whitelist definition

In this section, you can put as many entries as you want, even for the same type of filter. If at least one matches, the Map server can be used.

This section is not for defining which client can request maps. It is just here to avoid having the print module used as a proxy to access documents from computers behind firewalls.

There are 3 ways to whitelist a host.

Allowing every local services:

```
- !localMatch
  dummy: true
```

The `dummy` parameter is ignored, but mandatory to avoid a limitation in the YAML format.

Allowing by DNS name:

```
- !dnsMatch
  host: labs.metacarta.com
```

Allowing by IP address:

```
  - !ipMatch
    ip: www.camptocamp.org
?   mask: 255.255.255.255
```

The `ip` parameter can be a DNS name that will be resolved or directly an IP address.

All the methods accept the following optional parameters:

- port: to limit to a certain TCP port
- pathRegexp: a regexp that must match the path part of the URL (before the '?').

Metadata definition

Allow to add some metadata to the generated PDF. They are visible in acroread in the File->Properties menu.

The structure is like that:

```
    metaData:
?     title: ''
?     author: ''
?     subject: ''
?     keywords: ''
?     creator: ''
?     supportLegacyReader: false
```

All fields are optional and can use global variables, as defined in the Block definition chapter. Page specific variables are not accessible.

Page definition

The structure is like that:

```
      pageSize: A4
?     landscape: false
?     marginLeft: 40
?     marginRight: 40
```

```
?     marginTop: 20
?     marginBottom: 20
?     backgroundPdf: template.pdf
?     condition: null
?     header:
        height: 50
        items:
          - {BLOCK_DEFINITION}
          {...}
      items:
        - {BLOCK_DEFINITION}
        {...}
?     footer:
        height: 50
        items:
          - {BLOCK_DEFINITION}
          {...}
```

With the `condition` we can completely hide a page, same behavior than in block.

If `backgroundPdf` is specified, the first page of the given PDF file will be added as background of every page.

The `header` and `footer` sections are optional. If the `items` that are in the main section are too big, more pages are generated. The header and footer will be drawn on those pages as well.

Here is a short list of supported **pageSizes**:

name	width	height
LETTER	612	792
LEGAL	612	1008
A4	595	842
A3	842	1191

The complete list can be found in http://api.itextpdf.com/itext/com/itextpdf/text/PageSize.html. If you want to use a custom page size, you can set **pageSize** to the width and the height separated by a space.

Block definition

The next sub-sections document the possible types of blocks.

In general, text values or URLs can contain values taken from the **spec** structure coming with the client's request. A syntax similar to shell is used: ${variableName}. If the current page is a **titlePage**, only the root values are taken. If it's a **mainPage**, the service will first look in the current **page** section then in the root values. Here is how to use this functionality:

```
text: 'The value of mapTitle is: ${mapTitle}'
```

Some virtual variables can be used:

- ${pageNum}: The current page number.

- ${pageTot}: The total number of pages. Can be used only in text blocks.
- ${now}: The current date and time as defined by the machine's locale.
- ${now FORMAT}: The current date and time as defined by the FORMAT string. The syntax is here: http://java.sun.com/j2se/1.5.0/docs/api/java/text/SimpleDateFormat.html.
- ${configDir}: The absolute path to the directory of the configuration file.
- ${format PRINTF VAR}: Format the value of VAR using the provided PRINTF format (for example: %,d).

All the blocks can have a condition attribute that takes a spec attribute name. If the attribute name exists and is not equal to `false` or `0`, the block is drawn. Otherwise, it is ignored. An exclamation mark may precede the condition to invert it, exclamation mark is part of yaml syntax, than the expression should be in quotes.

Example: show text block only if in the spec the attribute name `showText` is given, is not equal to `false` and not equal to `0`:

```
- !text
  text: 'mytext'
  condition: showText
```

Text block

```
        - !text
?         font: Helvetica
?         fontSize: 12
?         fontEncoding: Cp1252
?         fontColor: black
?         spacingAfter: 0
?         align: left
?         vertAlign: middle
?         backgroundColor: #FFFFFF
          text: 'Blahblah'
```

Typical `fontEncoding` values are:

- Cp1250
- Cp1252
- Cp1257
- Identity-H (horizontal UTF-8)
- Identity-V (vertical UTF-8)
- MacRoman

The `font` must refer to a standard PDF font or a declared font.

Image block

```
        - !image
          maxWidth: 200
          maxHeight: 100
?         spacingAfter: 0
```

```
?         align: left
?         vertAlign: middle
          url: http://trac.mapfish.org/trac/mapfish/chrome/site/img/mapfish.png
```

Supported formats are PNG, GIF, Jpeg, Jpeg2000, BMP, WMF (vector), SVG and TIFF.

The original aspect ratio will be respected. The url can contain `${}` variables.

Columns block

```
        - !columns
?         config: {TABLE_CONFIG}
?         widths: [25,25,25,25]
?         backgroundColor: #FFFFFF
?         absoluteX: null
?         absoluteY: null
?         width: {PAGE_WIDTH}
?         spacingAfter: 0
?         nbColumns: -1
          items:
            - {BLOCK_DEFINITION}
            {...}
```

Can be called **!table** as well.

By default, the width of the columns will be equal.

Each item will be in its own column.

If the **absoluteX**, **absoluteY** and **width** are given, the columns block will be floating on top of the page at the specified position.

The **widths** attribute can be used to change the width of the columns (by default, they have the same width). It must contain one integer for each column. The width of a given column is *tableWidth*columnWeight/sum(columnWeight)*.

Every block type is allowed except for **map** if the column has an absolute position.

Look at <http://trac.mapfish.org/trac/mapfish/wiki/PrintModuleServer#Tableconfiguration to know how to specify the **config** field.

Map block

Allowed only within a **mainPage**.

```
        - !map
          width: {WIDTH}
          height: {HEIGHT}
?         name: map
?         spacingAfter: 0
?         align: left
?         vertAlign: middle
?         absoluteX: null
?         absoluteY: null
?         overviewMap: null
?         backgroundColor: #FFFFFF
```

width and **height** are mandatory. You can use variable substitution in this part, but if you do so, the browser won't receive the map size when it calls **info.json**. You'll have to **override mapfish.widgets.print.Base.configReceived** and set the map width and height of your layouts.

If the **absoluteX** and **absoluteY** are given, the map block will be floating on top of the page at the specified position.

The **name** is what will be displayed in the Acrobat's reader layer panel. The map layers will be displayed bellow it.

If **overviewMap** is specified, the map will be an overview of the extent augmented by the given factor. There are few cases to consider with map overviews:

1. If there is no overview overrides and no OL.Control.MapOverview, then all the layers will figure in the PDF map overview.
2. If there are overview overrides, the OL map overview control is ignored.
3. If there are no overview overrides and there is an OL.Control.MapOverview (takes the first one), then the layers defined in the control are taken into account. By default it is the current base layer.

Scalebar block

Display a scalebar.

Allowed only within a **mainPage**.

```
        - !scalebar
          maxSize: 150
?         type: line
?         intervals: 3
?         subIntervals: false
?         units: m
?         barSize: 5
?         lineWidth: 1
?         barDirection: up
?         textDirection: up
?         labelDistance: 3
?         font: Helvetica
?         fontSize: 12
?         fontColor: black
?         color: #000000
?         barBgColor: null
?         spacingAfter: 0
?         align: left
?         vertAlign: middle
?         backgroundColor: #FFFFFF
?         lockUnits: true
```

The scalebar, will adapt its width up to *maxSize* (includes the labels) in order to have a multiple of 1, 2 or 5 values at each graduation. For example:

- 0, 1, 2, ...
- 0, 2, 4, ...
- 0, 5, 10, ...
- 0, 10, 20, ...

The *barSize* is the thickness of the bar or the height of the tick marks on the line. The *lineWith* is for the thickness of the lines (or bar border).

Units can be any of:

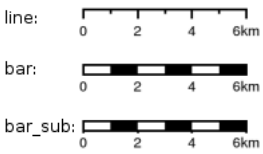
- m (mm, cm, m or km)
- ft (in, ft, yd, mi)
- degrees (min, sec, °)

If the value is too big or too small, the module will switch to one of the unit in parenthesis (the same unit is used for every intervals). If this behaviour is not desired, the lockUnits parameter will force the declared unit (or map unit if no unit is declared) to be used for the scalebar.

The number of *intervals* can be set to anything >=2. Labels are drawn only at main intervals. If there is no space to display a label at a certain interval, this label won't be displayed. If *subIntervals* are enabled, their number will depend on the length of an interval.

The type can be:

- line: A simple line with graduations
- bar: A thick bar with a suite of color and barBgColor blocks.
- bar_sub: Like bar, but with little lines for labels.



The bar and/or text orientation can be set to `up`, `down`, `left` or `right`.

The *align* attribute is for placing the whole scalebar withing the surrounding column or page. The *vertAlign* attribute is used only when placed in a column.

Labels are always centered on the graduation, at a distance specified by labelDistance.

Attributes block

Allows to display a table of the displayed feature's attributes.

Allowed only within a *mainPage*.

```
        - !attributes
          source: results
?         tableConfig: {TABLE_CONFIG}
          columnDefs:
            {COLUMN_NAME}:
?             columnWeight: 0        MF_V1.2
              header: {BLOCK_DEFINITION}
              cell: {BLOCK_DEFINITION}
            {...}
```

Look here for how to specify the *tableConfig* field.

The *columnWeigth* (MF_V1.2 only) allows to define a weight for the column width. If you specify it for one column, you have to specify it for all of them. The width of a given column is tableWidth*columnWeight/sum(columnWeight).

The **source** value defines the name of the entry in the root of the client's **spec**. For example, it would look like that:

```
{
  ...
  pages: [
    {
      ...
      results: {
        data: [
          {id:1, name: 'blah', icon: 'icon_pan'},
          ...
        ],
        columns: ['id', 'name', 'icon']
      }
    }
  ]
  ...
}
```

With this spec you would have to define 3 columnDefs with the names **id**, **name** and **icon**. Each cell definition blocks have access to all the values of the current row.

The spec part is filled automatically by the 2 MapFish widgets when their grids parameter is set.

Here is a crazy example of columnDef that will show the name of the icon and it's bitmap side-by-side inside a single column:

```
columnDefs:
  icon:
    header: !text
      text: Symbol
      backgroundColor: #A0A0A0
    cell: !columns
      items:
        - !text
          text: '${icon}'
        - !image
          align: center
          maxWidth: 15
          maxHeight: 15
          url: 'http://www.mapfish.org/svn/mapfish/trunk/MapFish/client/mfbase/
↪mapfish/img/${icon}.png'
```

A more complex example can be found in SVN: config.yaml spec.json

The print widgets are able to fill the spec for you based on a dictionary of **Ext.grid.GridPanel**. Just pass them through the grids parameter.

Legends block

Display each layers along with its classes (icons and labels).

```
        - !legends
?         backgroundColor: #FFFFFF
?         borders: false
?         horizontalAlignment: center
?         maxWidth: 0
```

```
?          maxHeight: 0
?          iconMaxWidth: 0
?          iconMaxHeight: 8
?          iconPadding: 8 7 6 5
?          textMaxWidth: 8
?          textMaxHeight: 8
?          textPadding: 8 7 6 5
?          defaultScale: 1.0
?          inline: true
?          classIndentation: 20
?          layerSpaceBefore: 5
?          layerSpace: 5
?          classSpace: 2
?          layerFont: Helvetica
?          layerFontSize: 10
?          classFont: Helvetica
?          classFontSize: 8
?          fontEncoding: Cp1252
?          columnMargin: 3
```

borders is mainly for debugging purpouses and shows all borders in the legend tables. This can be either 'true' or 'false'.

horizontalAlignment can be left, right or center (default) and aligns all items left, right or in the center.

iconMaxWidth, **iconMaxHeight**, **defaultScale** with value of 0 indicate that the value will be ignored, i.e. the values are automatically set to the equivalent of Infinity, Infinity and 1 respectively. If the legends URL passed to MapFish (see http://mapfish.org/doc/print/protocol.html#print-pdf) are obtained from a WMS GetLegendGraphic request, the width/height are only indicative (even more when a label text is included with *LEGEND_OPTIONS/forceLabels parameter*) and it would be safer, in order to preserve scale coherence between legends and map, to set **iconMaxWidth** and **iconMaxHeight** to zero.

textMaxWidth/Height and **iconMaxWidth/Height** define how wide/high the text/icon cells of a legend item can be. At this point textMaxHeight is ignored.

textPadding and **iconPadding** can be used like standard CSS padding. In the above example 8 is the padding top, 7 padding right, 6 padding bottom and 5 padding left.

if **inline** is true icons and text are rendered on the same line, BUT multicolumn is still enabled.

if **maxWidth** is set the whole legend gets a maximum width, just like other blocks. Note that **maxWidth** does not have any impact on icons size, thus icons may overflow outside the legends block.

if **maxHeight** is set the whole legend gets a maximum height. This forces more than one column to appear if the legend is higher than the specified value. This can be used to enable the multi-column layout. 0 makes the maxHeight= max value, i.e. the equivalent of infinity.

if **defaultScale** is non null it means that the legend image will be scaled so it doesn't take the full space. This can be overriden for individual classes in the spec JSON sent to the print module by adding an attribute called 'scale' and giving it a number. In conjunction with iconMaxWidth/Height this can be used to control average and also maximum width/height. If **defaultScale** equals 1, one pixel is scaled to one point (1/72 inch) in generated PDF. By default, as GeoServer legends are generated with ~90 dpi resolution (exactly 25.4/0.28), setting **defaultScale** value to 0.7937 (72*0.28/25.4) produces legend icons of same size as corresponding map icons. As the *LEGEND_OPTIONS/dpi GeoServer parameter* is not handled by MapFish, the resolution will necessary be ~91 dpi, which may cause visual quality difference with the map.

For this to work, you need to set the **layerTree** config option on MF print widgets, more precisely the legends should be present in the print.pdf JSON request.

layerSpaceBefore is to specify the space before the second and consecutive layers.

layerSpace and **classSpace** is to specify the line space to add after layers and classes.

columnMaxWidth maximum width of a column in multi-column layout. Not tested (at time of writing).

classIndentation amount of points to indent classes by.

layerSpaceBefore if a layer is after another one, this defines the amount of space to have before it. This will not be applied if the layer is the first item in its column in multi-column layout.

layerFont font of layer name legend items.

layerFontSize font size of layer name.

classFont font of class legend items.

classFontSize font size of class.

fontEncoding (see below)

Table configuration

The *columns block* and the *attributes block* can take a table configuration object like that:

```
    config:
?     borderWidth: 0
?     borderWidthLeft: 0
?     borderWidthRight: 0
?     borderWidthTop: 0
?     borderWidthBottom: 0
?     borderColor: black
?     borderColorLeft: black
?     borderColorRight: black
?     borderColorTop: black
?     borderColorBottom: black
?     cells:
?       - {CELL_CONFIGURATION}
```

A cell configuration looks like that:

```
?     row: {...}
?     col: {...}
?     borderWidth: 0
?     borderWidthLeft: 0
?     borderWidthRight: 0
?     borderWidthTop: 0
?     borderWidthBottom: 0
?     borderColor: black
?     borderColorLeft: black
?     borderColorRight: black
?     borderColorTop: black
?     borderColorBottom: black
?     padding: 0
?     paddingLeft: 0
?     paddingRight: 0
?     paddingTop: 0
?     paddingBottom: 0
?     backgroundColor: white
?     align: LEFT
?     vertAlign: TOP
```

The stuff configured at table level is for the table border, not every cell.

The **cells** list defines overrides for some cells. The cells an override is applied to is defined by the **row** and **col** attribute. Those attributes can have several formats:

- **0**: apply only to row or column 0 (the first)
- **0-10**: applies only the row or columns from 0 to 10
- or you can use any regular expression

Every matching overrides is applied in order and will override the values defined in the previous ones.

For example, if you want to draw an attribute block like that:

Seuchen grouppe	Tier
Auszurottende Seuchen	Ziege
Auszurottende Seuchen	Ziege
Zu bekämpfende Seuchen	Bienen
Zu bekämpfende Seuchen	Bienen
Zu bekämpfende Seuchen	Bienen

You define that:

```
- !attributes
  tableConfig:
    borderWidth: 1
    cells:
      # match every cell (default cell formatting)
      - borderWidthBottom: 0.5
        borderWidthLeft: 0.5
        padding: 4
        paddingTop: 0
      # match every even cell (yellowish background)
      - row: '\d*[02468]'
        backgroundColor: #FFFFCC
      # for the header
      - row: 0
        borderWidthBottom: 1
        backgroundColor: #FA0002
        align: center
  {...}
```

Warranty disclaimer and license

The authors provide these documents “AS-IS”, without warranty of any kind either expressed or implied.

Document under Creative Common License Attribution-Share Alike 2.5 Generic.

Authors: MapFish developers.

Protocol

Four commands are available and are documented in the next sections.

Every command uses the HTTP status code to notify errors.

info.json

HTTP command:

```
GET {PRINT_URL}/info.json?url={PRINT_URL}%2Finfo.json&var=printConfig
```

Returns a JSON structure as such:

```
var printConfig = {
    "scales":[
        {"name":"25000"},
        {"name":"50000"},
        {"name":"100000"}
    ],
    "dpis":[
        {"name":"190"},
        {"name":"254"}
    ],
    "outputFormats":[
        {"name":"pdf"},
        {"name":"png"}
    ],
    "layouts":[
        {
            "name":"A4 portrait",
            "map":{
                "width":440,
                "height":483
            }
        }
    ],
    "printURL":"http:\/\/localhost:5000\/print\/print.pdf",
    "createURL":"http:\/\/localhost:5000\/print\/create.json"
}
```

This can be loaded through an HTML script tag like that:

```
<script type="text/javascript"
      src="http://localhost:5000/print/info.json?var=printConfig"></script>
```

or through an AJAX request, in this case the `var` query parameter will be omitted.

The "url" query parameter is here to help the print servlet to know what URL is used by the browser to access the servlet. This parameter is here because the servlet can be behind a proxy, hiding the real URL.

print.pdf

HTTP command:

```
GET {PRINT_URL}/print.pdf?spec={SPEC}
   or
POST {PRINT_URL}/print.pdf    with {SPEC} in the request body
```

The "SPEC" parameter is a JSON structure like that:

```
{
    layout: 'A4 portrait',
```

```
    ...CUSTOM_PARAMS...
    srs: 'EPSG:4326',
    units: 'degrees',
    geodetic: false,
    outputFilename: 'political-boundaries',
    outputFormat: 'pdf',
    mergeableParams: {
        cql_filter: {
            defaultValue: 'INCLUDE',
            separator: ';',
            context: 'http://labs.metacarta.com/wms/vmap0'
        }
    },
    layers: [
        {
            type: 'WMS',
            layers: ['basic'],
            baseURL: 'http://labs.metacarta.com/wms/vmap0',
            format: 'image/jpeg'
        }
    ],
    pages: [
        {
            center: [6, 45.5],
            scale: 4000000,
            dpi: 190,
            geodetic: false,
            strictEpsg4326: false,
            ...CUSTOM_PARAMS...
        }
    ],
    legends: [
        {
            classes: [
                {
                    icons: [
                        'full url to the image'
                    ],
                    name: 'an icon name',
                    iconBeforeName: true
                }
            ],
            name: 'a class name'
        }
    ]
}
```

The location to show on the map can be specified with a **center** and a **scale** as show or with a **bbox** like that:

```
bbox: [5, 45, 6, 46]
```

The print module will use the nearest scale and will make sure the aspect ratio stays correct.

The geodetic parameter can be set to true so the scale of geodetic layers can correctly be calculated. Certain projections (Google and Latlong for example) are based on a spheroid and therefore require **geodetic: true** in order to correctly calculate the scale. If the geodetic parameter is not present it will be assumed to be false.

The _optional_ strictEpsg4326 parameter can be set to true to control how EPSG:4326 is interpreted. This needs to be true for WMS version 1.3.0 GetMap requests. See https://www.google.ch/search?q=epsg+4326+latitude+longitude+order&oq=epsg+4326+&aqs=chrome.3.69i57j0l5.5996j0j8 for some links to the history and mess that is EPSG:4326.

The outputFilename parameter is optional and if omitted the values used in the server's configuration will be used instead. If it is present it will be the name of the downloaded file. The suffix will be added if not left off in the parameter. The date can be substituted into the filename as well if desired. See configuration's outputFilename for more information and examples

The outputFormat parameter is optional and if omitted the value 'pdf' will be used. Only the formats returned in the info are permitted.

There are two locations where custom parameters can be added. Those will be ignored by the web service but, will be accessible from the layout templates.

Some layer types support merging more layers request into one, when the server is the same (for example WMS). For those, a mergeableParams section can be used to define merging strategies for some custom parameters. The default rule is to merge layers with identical custom parameters. Using mergeableParams, defined parameters values can be joined using a given separator and a default value if some of the layers miss the parameter. Mergeable parameters can have a context, that is the baseURL they can be used for (if not defined they will be used for every layer).

For the format of the **layers** section, please look at the implementations pointed by mapfish.PrintProtocol.SUPPORTED_TYPES.

This command returns the PDF file directly.

create.json

HTTP command:

```
POST {PRINT_URL}/create.json?url={PRINT_URL}%2Fcreate.json
```

The spec defined in the "print.pdf" command must be included in the POST body.

Returns a JSON structure like that:

```
{
    getURL: 'http:\/\/localhost:5000\/print\/56723.pdf'
}
```

The URL returned can be used to retrieve the PDF file. See the next section.

{ID}.pdf

This command's URL is returned by the "create.json" command.

HTTP command:

```
GET {PRINT_URL}/{ID}.pdf
```

Returns the PDF. Can be called only during a limited time since the server side temporary file is deleted afterwards.

Multiple maps on a single page

To print more than one map on a single page you need to:

- specify several map blocks in a page of the yaml file, each with a distinct name property value
- use a particular syntax in the spec to bind different rendering properties to each map block

This is possible specifying a _maps_ object in spec root object with a distinct key - object pair for each map. The key will refer the map block name as defined in yaml file. The object will contain layers and srs for the named map. Another _maps_ object has to be specified inside the page object to describe positioning, scale and so on.

```
{
    ...
    maps: {
        "main": {
            layers: [
                ...
            ],
            srs: 'EPSG:4326'
        },
        "other": {
            layers: [
                ...
            ],
            srs: 'EPSG:4326'
        }
    },
    ...
    pages: [
        {
            maps: {
                "main": {
                    center: [6, 45.5],
                    scale: 4000000,
                    dpi: 190,
                    geodetic: false,
                    strictEpsg4326: false,
                    ...CUSTOM_PARAMS...
                },
                "other": {
                    center: [7.2, 38.6],
                    scale: 1000000,
                    dpi: 300,
                    geodetic: false,
                    strictEpsg4326: false,
                    ...CUSTOM_PARAMS...
                }
            }

        }
    ],
    ...
}
```

Other config blocks have been enabled to multiple maps usage. The scalebar block can be bound to a specific map, specifying a name property that matches the map name. Also, in text blocks you can use the ${scale.<mapname>} placeholder to print the scale of the map whose name is <mapname>.

Layers Params

Vector

Type: vector

Render vector layers. The geometries and the styling comes directly from the spec JSON.

- opacity (Defaults to `1.0`)
- geoJson (Required) the geoJson to render
- styleProperty (Defaults to '_style') Name of the property within the features to use as style name. The given property may contain a style object directly.
- styles (Optional) dictonary of styles. One style is defined as in OpenLayers.Feature.Vector.style.
- name (Defaults to `vector`) the layer name.

WMS

Type: wms

Support for the WMS protocol with possibilities to go through a WMS-C service (TileCache).

- opacity (Defaults to `1.0`)
- baseURL (Required) Service URL
- customParams (Optional) Map, additional URL arguments
- layers (Required)
- styles (Optional)
- format (Required)
- version (Defaults to `1.1.1`)
- useNativeAngle (Defaults to false) it true transform the map angle to customParams.angle for GeoServer, and customParams.map_angle for MapServer.

WMTS

Type: wmts

Support for the protocol using directly the content of a WMTS tiled layer, support REST or KVP.

Two possible mode, standard or simple, the simple mode imply that all the topLeftCorner are identical.

Standard mode:

- opacity (Defaults to 1.0)
- baseURL the 'ResourceURL' available in the WMTS capabilities.
- customParams (Optional) Map, additional URL arguments
- layer (Required) the layer name
- version (Defaults to `1.0.0`) WMTS protocol version
- requestEncoding (Defaults to `REST`) `REST` or `KVP`

- style (Optional) the style name
- dimensions (Optional) list of dimensions names
- params (Optional) dictionary of dimensions name (capital) => value
- matrixSet (Required) the name of the matrix set
- matrixIds (Required) array of matrix ids e.g.:

```
[{
    "identifier": "0",
    "matrixSize": [1, 1],
    "resolution": 4000,
    "tileSize": [256, 256],
    "topLeftCorner": [420000, 350000]
}, ...]
```

- format (Optional, Required id requestEncoding is `KVP`)

Simple mode:

- baseURL base URL without the version.
- layer (Required)
- version (Required)
- requestEncoding (Required) `REST`
- tileOrigin (Required)
- tileSize (Required)
- extension (Required)
- resolutions (Required)
- style (Required)
- tileFullExtent (Required)
- zoomOffset (Required)
- dimensions (Optional)
- params (Optional)
- formatSuffix (Required)

Tms

Type: tms

Support the TMS tile layout.

- opacity (Defaults to 1.0)
- baseURL (Required) Service URL
- customParams (Optional) Map, additional URL arguments
- maxExtent (Required) Array, extent coordinates `[420000, 30000, 900000, 350000]`
- tileSize (Required) Array, tile size e.g. `[256, 256]`
- format (Required)

- layer (Required)
- resolutions (Required) Array of resolutions
- tileOrigin (Optional) Object, tile origin. Defaults to `0,0`

Resources:

- Quick intro to TMS requests: http://geowebcache.org/docs/current/services/tms.html
- TMS Spec (Not an Official Standard): http://wiki.osgeo.org/wiki/Tile_Map_Service_Specification

Xyz

Type: xyz

Support the tile layout z/x/y.<extension>.

- opacity (Defaults to 1.0)
- baseURL (Required) Service URL
- customParams (Optional) Map, additional URL arguments
- maxExtent (Required) Array, extent coordinates `[420000, 30000, 900000, 350000]`
- tileSize (Required) Array, tile size e.g. `[256, 256]`
- resolutions (Required) Array of resolutions (Required) Array of resolutions
- extension (Required) file extension (Required) file extension
- tileOrigin (Optional) Array, tile origine e.g. `[420000, 350000]`
- tileOriginCorner `tl` or `bl` (Defaults to `bl`)
- path_format (Optional) url fragment used to construct the tile location. Can support variable replacement of `${x}`, `${y}`, `${z}` and `${extension}`. Defaults to zz/x/y.extension format. You can use multiple "letters" to indicate a replacable pattern (aka, `${zzzz}` will ensure the z variable is 0 padded to have a length of AT LEAST 4 characters).

Osm

Type: osm

Support the OSM tile layout.

- opacity (Defaults to `1.0`)
- baseURL (Required) Service URL
- customParams (Optional) Map, additional URL arguments
- maxExtent (Required) Array, extent coordinates `[420000, 30000, 900000, 350000]`
- tileSize (Required) Array, tile size e.g. `[256, 256]`
- resolutions (Required) Array of resolutions
- extension (Required) file extension

TileCache

Type: tileCache

Support for the protocol using directly the content of a TileCache directory.

- opacity (Defaults to `1.0`)
- baseURL (Required) Service URL
- customParams (Optional) Map, additional URL arguments
- layer (Required)
- maxExtent (Required) Array, extent coordinates `[420000, 30000, 900000, 350000]`
- tileSize (Required) Array, tile size e.g. `[256, 256]`
- resolutions (Required) Array of resolutions
- extension (Required) file extension

Image

Type: image

- opacity (Defaults to `1.0`)
- name (Required)
- baseURL (Required) Service URL
- extent (Required)

MapServer

Type: mapServer

Support mapserver WMS server.

- opacity (Defaults to `1.0`)
- baseURL (Required) Service URL
- customParams (Optional) Map, additional URL arguments
- layers (Required)
- format (Required)

KaMap

Type: kaMap

Support for the protocol using the KaMap tiling method

- opacity (Defaults to `1.0`)
- baseURL (Required) Service URL
- customParams (Optional) Map, additional URL arguments
- map

- group
- maxExtent (Required) Array, extent coordinates `[420000, 30000, 900000, 350000]`
- tileSize (Required) Array, tile size e.g. `[256, 256]`
- resolutions (Required) Array of resolutions
- extension (Required) file extension

KaMapCache

Type: kaMapCache

Support for the protocol talking direclty to a web-accessible ka-Map cache generated by the precache2.php script.

- opacity (Defaults to `1.0`)
- baseURL (Required) Service URL
- customParams (Optional) Map, additional URL arguments
- map (Required)
- group (Required)
- metaTileWidth (Required)
- metaTileHeight (Required)
- units (Required)
- maxExtent (Required) Array, extent coordinates `[420000, 30000, 900000, 350000]`
- tileSize (Required) Array, tile size e.g. `[256, 256]`
- resolutions (Required) Array of resolutions
- extension (Required) file extension

Google

Type: google or tiledGoogle

They used the Google Map Static API, tiledGoogle will create tiles and google only one image.

The google map reader has several custom parameters that can be added to the request they are:

- opacity (Optional, Defaults to `1.0`)
- baseURL (Required, should be 'http://maps.google.com/maps/api/staticmap')
- customParams (Optional) Map, additional URL arguments
- maxExtent (Required, should be `[-20037508.34, -20037508.34, 20037508.34, 20037508.34]`)
- resolutions (Required, should be `[156543.03390625, 78271.516953125, 39135.7584765625, 19567.87923828125, 9783.939619140625, 4891.9698095703125, 2445.9849047851562, 1222.9924523925781, 611.4962261962891, 305.74811309814453, 152.87405654907226, 76.43702827453613, 38.218514137268066, 19.109257068634033, 9.554628534317017, 4.777314267158508,`

```
2.388657133579254, 1.194328566789627, 0.5971642833948135, 0.
29858214169740677, 0.14929107084870338, 0.074645535424351691])
```

- extension (Required, should be `png`)
- client (Optional)
- format (Optional)
- maptype (Required) - type of map to display: http://code.google.com/apis/maps/documentation/staticmaps/#MapTypes
- sensor (Optional) - specifies whether the application requesting the static map is using a sensor to determine the user's location
- language (Optional) - language of labels.
- markers (Optional) - add markers to the map: http://code.google.com/apis/maps/documentation/staticmaps/#Markers

```
markers: ['color:blue|label:S|46.5195933305192,6.566684726913701']
```

- path (Optional) - add a path to the map: http://code.google.com/apis/maps/documentation/staticmaps/#Paths

```
path: 'color:0x0000ff|weight:5|46.5095933305192,6.506684726913701|46.5195933305192,6.
↪526684726913701|46.5395933305192,6.536684726913701|46.5695933305192,6.
↪576684726913701',
```

Warranty disclaimer and license

The authors provide these documents "AS-IS", without warranty of any kind either expressed or implied.

Document under Creative Common License Attribution-Share Alike 2.5 Generic.

Authors: MapFish developers.

FAQ

All I get in my PDF is: "ERROR: infinite table loop". What's wrong? Something in your page is too big. For example, the width or the height of your !map block.

I tried to print (pylons mode) and I get a "Java error". What's next? Look in the apache error log, you'll find more information.

What are the limitations of the type 2 layers? It depends mostly on the map server you use. For the moment, GeoServer has not been extensively tested. With MapServer:

- The PDF output must be enabled when you compile and doesn't work in WMS mode, only in native MapServer mode. There are some limitations. on the styling. And you must use truetype fonts.
- The SVG output is limited regarding the stylings you can use. For example only plain polygon fillings are supported by MapServer. If a complex styling is used, your features may appear plain black.

I tried to change the layout and half the Map is printed off the page on the right. Or I have an empty page added. Is it a It's mostly a feature ;-) . This kind of behavior can be seen in iText, when adding a block that is too big for the page size. Try to reduce the size of your map block.

When I look at my generated PDF in Acrobat Reader, it looks good. But, when I print it, it misses some tiles/layers, som
There are three possible explanations:

- Your printer has not enough memory: in Acrobat's print dialog, select "Save printer memory"
- Your printer firmware is buggy: upgrade it
- Your printer driver is buggy: upgrade it

The module needs to go through a proxy to acces the map services. It's so 90s... you should hire some fresh guys for your IT team. ;-)

You need to set some system properties (http.proxy*) when you start your java programs.

On the browser, the scale is displayed with spaces to separate thousands and it's against my religion. How do I put my
By default, the browser's configured locale is used. You can force another locale in the print widget configuration:

```
{
  ...
  configUrl: 'print/info.json',
  serviceParams: { locale: 'fr_CH' },
  ...
}
```

I copied the examples and the print widgets are not working. First edit the client/examples/examples.js file and make sure the URLs are correct.

1. If you don't want to install the server side, make sure you installed a proxy (see Configure Proxy). For example, test (must return a JSON content, not the proxy.cgi script's content) it with an URL like that (adapt the hostname, port and path): *http://localhost/cgi-bin/proxy.cgi?url=http://demo.mapfish.org/mapfishsample/trunk/print/info.json*
2. If you installed the server side, make sure it works by calling the URL specified in the mapfish.SERVER_BASE_URL variable (must be the hostname/port your page is accessed through) added with */print/info.json*. For example, if you have *mapfish.SERVER_BASE_URL="http://localhost/mapfish": http://localhost/mapfish/print/info.json*

If it still doesn't work, use firefox, install firebug and check in the console panel that the AJAX request made by the print widget works fine.

Warranty disclaimer and license

The authors provide these documents "AS-IS", without warranty of any kind either expressed or implied.

Document under Creative Common License Attribution-Share Alike 2.5 Generic.

Authors: MapFish developers.

15.16 Cross-layer filtering

Cross-layer filtering provides the ability to find features from layer A that have a certain relationship to features in layer B. This can be used, for example, to find all bus stops within a given distance from a specified shop, or to find all coffee shops contained in a specified city district.

The **querylayer** module adds filter functions that implement cross-layer filtering. The functions work by querying a secondary layer within a filter being applied to a primary layer. The name of the secondary layer and an attribute to extract from it are provided as arguments, along with an ECQL filter expression to

determine which features are of interest. A common use case is to extract a geometry-valued attribute, and then use the value(s) in a spatial predicate against a geometry attribute in the primary layer.

Filter functions are widely supported in GeoServer, so cross-layer filtering can be used in SLD rules and WMS and WFS requests, in either XML or CQL filters.

15.16.1 Installing the querylayer module

1. Download the **querylayer** extension corresponding to your version of GeoServer.

   Warning: The version of the extension **must** match the version of the GeoServer instance

2. Extract the contents of the extension archive into the `WEB-INF/lib` directory of the GeoServer installation.
3. To check the module is properly installed request the WFS 1.1 capabilities from the GeoServer home page. The `Filter_Capabilities` section should contain a reference to a function named `queryCollection`.

```
...
<ogc:Filter_Capabilities>
    ...
    <ogc:ArithmeticOperators>
      ...
      <ogc:Functions>
        <ogc:FunctionNames>
          ...
          <ogc:FunctionName nArgs="-1">queryCollection</ogc:FunctionName>
          <ogc:FunctionName nArgs="-1">querySingle</ogc:FunctionName>
          ...
        </ogc:FunctionNames>
      </ogc:Functions>
    </ogc:ArithmeticOperators>
  </ogc:Scalar_Capabilities>
  ...
</ogc:Filter_Capabilities>
...
```

15.16.2 Function reference

The extension provides the following filter functions to support cross-layer filtering.

Name	Arguments	Description
querySingle	`layer` : String, `attribute` : String, `filter` : String	Queries the specified `layer` applying the specified *ECQL* `filter` and returns the value of `attribute` from the first feature in the result set. The layer name must be qualified (e.g. `topp:states`). If no filtering is desired use the filter `INCLUDE`.
queryCollection	`layer` : String, `attribute` : String, `filter` : String	Queries the specified `layer` applying the specified *ECQL* `filter` and returns a list containing the value of `attribute` for every feature in the result set. The layer name must be qualified (e.g. `topp:states`). If no filtering is desired use the filter `INCLUDE`. An exception is thrown if too many results are collected (see *Memory Limits*).
collectGeometries	`geometries`: a list of Geometry objects	Converts a list of geometries into a single Geometry object. The output of `queryCollection` must be converted by this function in order to use it in spatial filter expressions (since geometry lists cannot be used directly). An exception is thrown if too many coordinates are collected (see *Memory Limits*).

15.16.3 Optimizing performance

In the GeoServer 2.1.x series, in order to have cross-layer filters execute with optimal performance it is necessary to specify the following system variable when starting the JVM:

```
-Dorg.geotools.filter.function.simplify=true
```

This ensures the functions are evaluated once per query, instead of once per result feature. This flag is not necessary for the GeoServer 2.2.x series. (Hopefully this behavior will become the default in 2.1.x as well.)

15.16.4 Memory limits

The `queryCollection` and `collectGeometries` functions do not perform a true database-style join. Instead they execute a query against the secondary layer every time they are executed, and load the entire result into memory. The functions thus risk using excessive server memory if the query result set is very large, or if the collected geometries are very large. To prevent impacting server stability there are built-in limits to how much data can be processed:

- at most 1000 features are collected by `queryCollection`
- at most 37000 coordinates (1MB worth of Coordinate objects) are collected by `collectGeometries`

These limits can be overridden by setting alternate values for the following parameters (this can be done using JVM system variables, servlet context variables, or enviroment variables):

- `QUERY_LAYER_MAX_FEATURES` controls the maximum number of features collected by `queryCollection`
- `GEOMETRY_COLLECT_MAX_COORDINATES` controls the maximum number of coordinates collected by `collectGeometries`

15.16.5 WMS Examples

The following examples use the `sf:bugsites`, `sf:roads` and `sf:restricted` demo layers available in the standard GeoServer download.

- **Display only the bug sites overlapping the restricted area whose category is 3**:

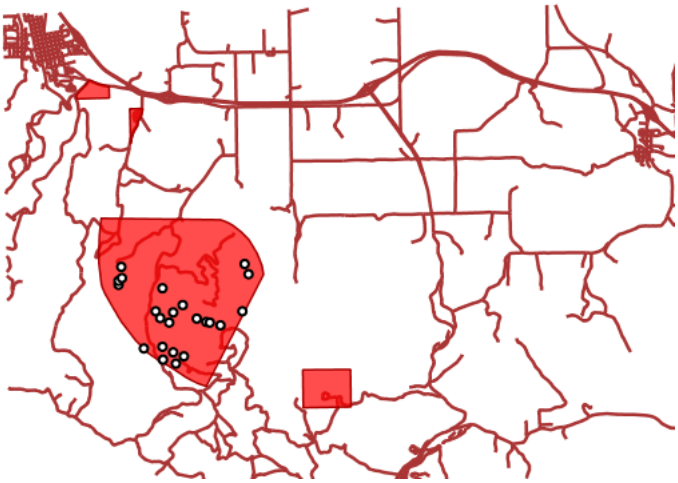
The CQL cross-layer filter on the `bugsites` layer is

```
INTERSECTS(the_geom, querySingle('restricted', 'the_geom','cat = 3')).
```

The WMS request is:

```
http://localhost:8080/geoserver/wms?LAYERS=sf%3Aroads%2Csf%3Arestricted%2Csf
↪%3Abugsites&STYLES=&FORMAT=image%2Fpng&SERVICE=WMS&VERSION=1.1.1&REQUEST=GetMap&
↪EXCEPTIONS=application%2Fvnd.ogc.se_inimage&SRS=EPSG%3A26713&CQL_FILTER=INCLUDE
↪%3BINCLUDE%3BINTERSECTS(the_geom%2C%20querySingle(%27restricted%27%2C%20%27the_geom
↪%27%2C%27cat%20%3D%203%27))&BBOX=589081.6705629,4914128.1213261,609174.02430924,
↪4928177.0717971&WIDTH=512&HEIGHT=358
```

The result is:



- **Display all bug sites within 200 meters of any road**:

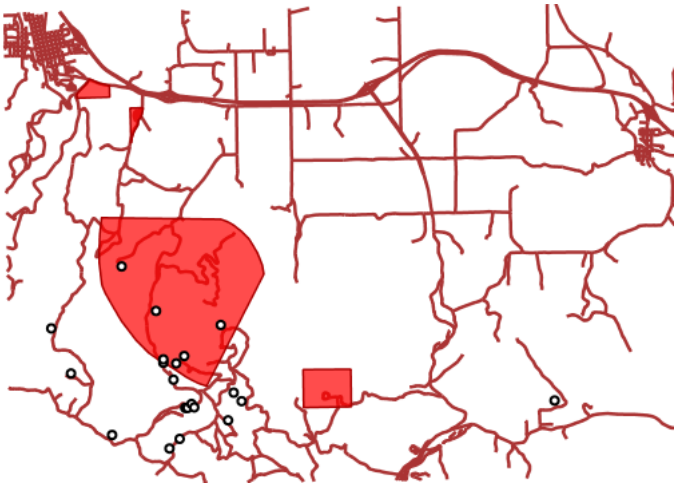
The CQL cross-layer filter on the `bugsites` layer is

```
DWITHIN(the_geom, collectGeometries(queryCollection('sf:roads',
'the_geom','INCLUDE')), 200, meters).
```

The WMS request is:

```
http://localhost:8080/geoserver/wms?LAYERS=sf%3Aroads%2Csf%3Arestricted%2Csf
↪%3Abugsites&STYLES=&FORMAT=image%2Fpng&SERVICE=WMS&VERSION=1.1.1&REQUEST=GetMap&
↪EXCEPTIONS=application%2Fvnd.ogc.se_inimage&SRS=EPSG%3A26713&CQL_FILTER=INCLUDE
↪%3BINCLUDE%3BDWITHIN(the_geom%2C%20collectGeometries(queryCollection(%27sf%3Aroads
↪%27%2C%27the_geom%27%2C%27INCLUDE%27))%2C%20200%2C%20meters)&BBOX=589042.42768447,
↪4914010.3926913,609134.78143081,4928059.3431623&WIDTH=512&HEIGHT=358
```

The result is:



15.16.6 WFS Examples

The following examples use the `sf:bugsites`, `sf:roads` and `sf:restricted` demo layers available in the standard GeoServer download.

- **Retrieve only the bug sites overlapping the restricted area whose category is 3**:

```
<wfs:GetFeature xmlns:wfs="http://www.opengis.net/wfs"
                xmlns:sf="http://www.openplans.org/spearfish"
                xmlns:ogc="http://www.opengis.net/ogc"
                service="WFS" version="1.0.0">
  <wfs:Query typeName="sf:bugsites">
    <ogc:Filter>
      <ogc:Intersects>
        <ogc:PropertyName>the_geom</ogc:PropertyName>
        <ogc:Function name="querySingle">
           <ogc:Literal>sf:restricted</ogc:Literal>
           <ogc:Literal>the_geom</ogc:Literal>
           <ogc:Literal>cat = 3</ogc:Literal>
        </ogc:Function>
      </ogc:Intersects>
    </ogc:Filter>
  </wfs:Query>
</wfs:GetFeature>
```

- **Retrieve all bugsites within 200 meters of any road**:

```
<wfs:GetFeature xmlns:wfs="http://www.opengis.net/wfs"
  xmlns:sf="http://www.openplans.org/spearfish"
  xmlns:ogc="http://www.opengis.net/ogc"
  service="WFS" version="1.0.0">
  <wfs:Query typeName="sf:bugsites">
    <ogc:Filter>
      <ogc:DWithin>
        <ogc:PropertyName>the_geom</ogc:PropertyName>
        <ogc:Function name="collectGeometries">
          <ogc:Function name="queryCollection">
            <ogc:Literal>sf:roads</ogc:Literal>
            <ogc:Literal>the_geom</ogc:Literal>
            <ogc:Literal>INCLUDE</ogc:Literal>
          </ogc:Function>
```

```
            </ogc:Function>
            <ogc:Distance units="meter">100</ogc:Distance>
          </ogc:DWithin>
        </ogc:Filter>
      </wfs:Query>
    </wfs:GetFeature>
```

15.17 Vector Tiles

GeoServer supports "vector tile" output in addition to the more standard image tile output. While the standard WMS output will generate a georeferenced map image, a vector tile contains georeferenced vector data, clipped into tiles for easy retrieval.

Note: This is an output format, not a data source.

Note: Here are two background video talks about "Vector Tiles with GeoServer and OpenLayers":

- FOSS4G.NA 2016
- FOSS4G 2016

15.17.1 Installing the Vector Tiles Extension

The Vector Tiles extension is an official extension, available on the GeoServer download page.

1. Download the extension for your version of GeoServer.

   Warning: Make sure to match the version of the extension to the version of GeoServer.

2. Extract the archive and copy the contents into the GeoServer `WEB-INF/lib` directory.
3. Restart GeoServer.

To verify that the extension was installed successfully

1. Open the *Web administration interface*
2. Click *Layers* and select a vector layer
3. Click the *Tile Caching* tab
4. Scroll down to the section on *Tile Formats*. In addition to the standard GIF/PNG/JPEG formats, you should see the following:
   - `application/json;type=geojson`
   - `application/json;type=topojson`
   - `application/x-protobuf;type=mapbox-vector`

   If you don't see these options, the extension did not install correctly.

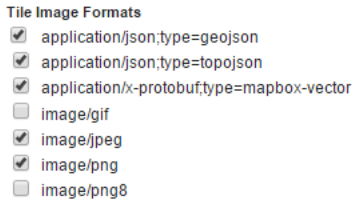


Fig. 15.37: Vector tiles tile formats

15.17.2 Vector tiles tutorial

This tutorial will show how to use the GeoServer vector tiles output.

Why use vector tiles?

The advantages of vector tiles are;

- **Rendering is done by the client** (for example, OpenLayers), not by the server. This allows different maps/applications to style a map differently without having to reconfigure GeoServer.
- **The size of a vector tile is usually smaller** than an image tile, resulting in faster data transfer and lower bandwidth usage.
- GeoWebCache, embedded with GeoServer efficiently stores the vector tile data. Since styling is done by the client, not the server, **GeoWebCache only needs to store one tile for all different styles**.
- Because the vector data is available on the client, very **high-resolution maps can be drawn without corresponding increases in bandwidth**.
- **The client has native access to the actual feature information** (attributes and geometry), allowing for very sophisticated rendering.

On the other hand, the main disadvantage of vector tiles is that the geographic data may need to be pre-processed to allow the client to do the drawings it requires (similar to preprocessing data for image maps). With this in mind, **vector tiles should only be used for rendering**.

Vector tile formats

GeoServer can also produce vector tiles in three formats: GeoJSON, TopoJSON, and MapBox Vector (MVT). These are also supported by OpenLayers and other clients.

Warning: When using vector tiles, be sure to use an up-to-date client. Older clients do not support all vector tiles capabilites and may result in rendering errors. We recommend using the latest version of OpenLayers (Currently v5.3.0).

- MVT is the preferred format for production.

Format	MIME	Description
MapBox Vector (MVT)	`application/x-protobuf;type=mapbox-vector`	**Recommended Format** This is an efficient binary format that is widely supported by almost all Vector Tile applications.
GeoJSON	`application/json;type=geojson`	This is a human readable JSON format. Although many geo-spatial applications support GeoJSON datasets, few Vector Tile applications support tiles in this format. Supported by Open Layers 3.
TopoJSON	`application/json;type=topojson`	This is a very complex, but somewhat human readable JSON format that is good for polygon coverages. It is not a widely supported and very few Vector Tile applications support it. Suported by Open Layers 3.

Publish vector tiles in GeoWebCache

We'll be publishing our vector tiles through GeoWebCache and publishing the layer in a custom OpenLayers application.

For this tutorial, we'll be using the layer `opengeo:countries` to show off the capabilities, though with slight modifications, any layer will do.

Note: Download the Admin 0 - Countries shapefile and publish the layer as `opengeo:countries`.

1. In the GeoServer admin interface, click `Tile Layers` under `Tile Caching`.

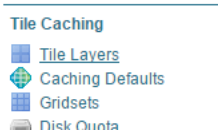


Fig. 15.38: Tile Layers

2. Click `opengeo:countries` in the list of layers.
3. By default the tile formats are `image/jpeg` and `image/png`. Check the boxes for the following vector tile formats:
   - `application/json;type=geojson`
   - `application/json;type=topojson`
   - `application/x-protobuf;type=mapbox-vector`

Fig. 15.39: Vector tiles tile formats

4. Click `Save`.

Our layer is now ready to be served.

Create OpenLayers application

1. Create a `www/vectortiles` directory inside your GeoServer Data Directory.
2. Download the latest version of OpenLayers.
3. Extract the following files to from the downloaded archive to the directory created in step 1:
   - `ol.js`
   - `ol-debug.js`
   - `ol.css`
4. In a text editor, create a new file with the following content:

```
<!DOCTYPE html -->
<html>
<head>
  <title>Vector tiles</title>
  <script src="ol.js"></script>
  <link rel="stylesheet" href="ol.css">
  <style>
    html, body {
      font-family: sans-serif;
      width: 100%;
    }
    .map {
      height: 500px;
      width: 100%;
    }
  </style>
</head>
<body>
  <h3>Mapbox Protobuf - vector tiles</h3>
  <div id="map" class="map"></div>
  <script>

  var style_simple = new ol.style.Style({
    fill: new ol.style.Fill({
      color: '#ADD8E6'
    }),
    stroke: new ol.style.Stroke({
      color: '#880000',
      width: 1
    })
  });

  function simpleStyle(feature) {
    return style_simple;
  }

  var layer = 'opengeo:countries';
  var projection_epsg_no = '900913';
  var map = new ol.Map({
    target: 'map',
```

```
    view: new ol.View({
      center: [0, 0],
      zoom: 2
    }),
    layers: [new ol.layer.VectorTile({
      style:simpleStyle,
      source: new ol.source.VectorTile({
        tilePixelRatio: 1, // oversampling when > 1
        tileGrid: ol.tilegrid.createXYZ({maxZoom: 19}),
        format: new ol.format.MVT(),
        url: '/geoserver/gwc/service/tms/1.0.0/' + layer +
            '@EPSG%3A'+projection_epsg_no+'@pbf/{z}/{x}/{-y}.pbf'
      })
    })]
  });
  </script>
</body>
</html>
```

5. Save this file in the directory created above as `index.html`.
6. Navigate to `http://localhost:8080/geoserver/www/vectortiles/index.html` and verify that the output shows without any errors.

Note: If your GeoServer is deployed at a server other than `http://localhost:8080/geoserver/`, then please adjust the above URL.

Fig. 15.40: Vector tile output

These tiles are being rendered by the OpenLayers client.

Styling vector tiles

Since these tiles are rendered in the client, we need only change the styling instructions inside the client application. No changes to GeoServer are required, and tiles will not have to be regenerated.

1. Change the fill color to light green:

```
var style_simple = new ol.style.Style({
  fill: new ol.style.Fill({
    color: 'lightgreen'
  }),
   stroke: new ol.style.Stroke({
      color: '#880000',
      width: 1
    })
}) ;
```

2. Save the file and reload the application.

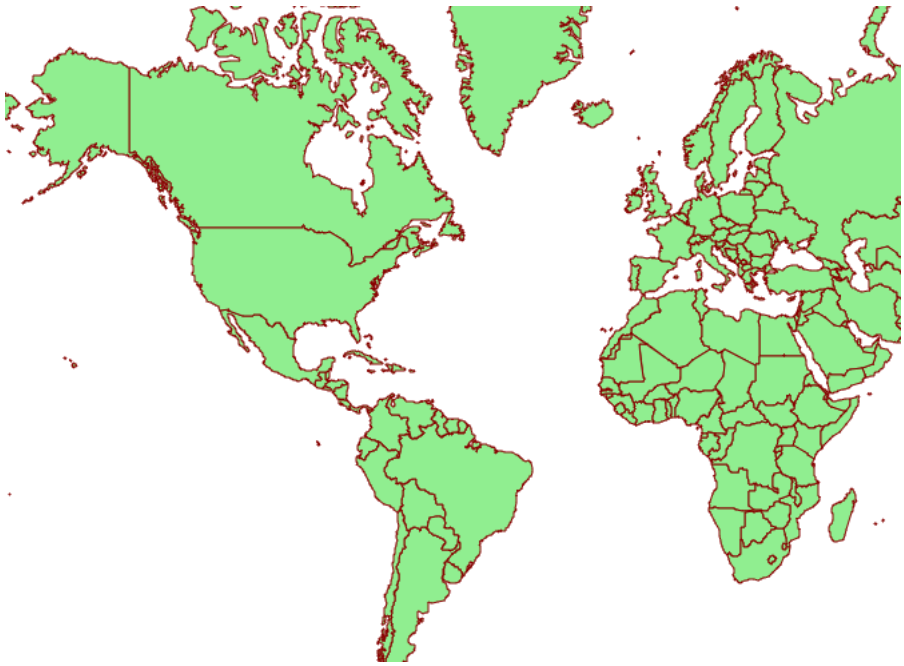


Fig. 15.41: Vector tile output with alternate color

3. We can also do attributed-based styling. This dataset contains has a property (`region_un`) which contains the region the country is in. Let's highlight countries in Africa by adding another style definition below the existing style:

```
var style_highlighted = new ol.style.Style({
  fill: new ol.style.Fill({
    color: 'yellow'
  }),
  stroke: new ol.style.Stroke({
    color: '#880000',
    width: 1
  })
});
```

4. Replace the existing style function:

```
function simpleStyle(feature) {
  return style_simple;
}
```

with the following:

```
function simpleStyle(feature) {
  if (feature.get("region_un") == "Africa") {
    return style_highlighted;
  }
  return style_simple;
}
```

5. Save the file and reload the application.

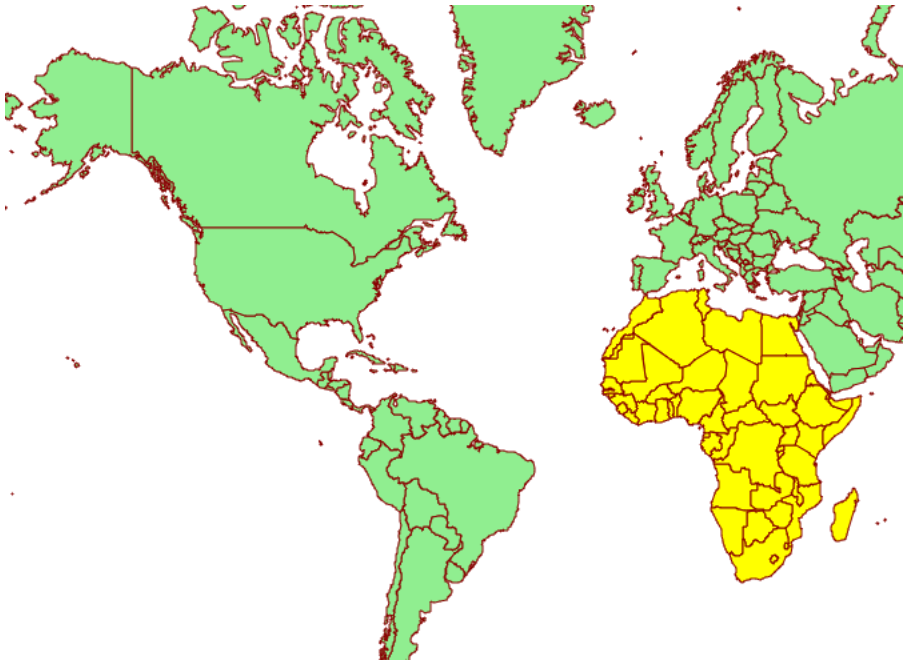


Fig. 15.42: Vector tile output with Africa highlighted

15.18 XSLT WFS output format module

The `xslt` module for GeoServer is a WFS output format generator which brings togheter a base output, such as GML, and a XSLT 1.0 style sheet to generate a new textual output format of user choosing.

The configuration for this output format can be either performed directly on disk, or via a REST API.

15.18.1 Manual configuration

All the configuration for the output resides in the `$GEOSERVER_DATA_DIR/wfs/transform` folder, which is going to be created on startup when missing if the XSLT output format has been installed in GeoServer.

Each XSLT transformation must be configured with its own xml file in the `$GEOSERVER_DATA_DIR/wfs/transform` folder, which in turn points to a xslt file for the transformation. While the names can be freeform, it is suggested to follow a simple naming convention:

- <mytransformation>.xml for the xml config file
- <mytransformation>.xslt for the xslt tile

Transformations can be either global, and thus applicable to any feature type, or type specific, in which case the transformation knows about the specific attributes of the transformed feature type.

15.18.2 Global transformation example

Here is an example of a global transformation setup. The `$GEOSERVER_DATA_DIR/wfs/transform/global.xml` file will look like:

```
<transform>
  <sourceFormat>text/xml; subtype=gml/2.1.2</sourceFormat>
  <outputFormat>HTML</outputFormat>
  <outputMimeType>text/html</outputMimeType>
  <fileExtension>html</fileExtension>
  <xslt>global.xslt</xslt>
</transform>
```

Here is an explanation of each element:

- `sourceFormat` (mandatory): the output format used as the source of the XSLT transformation
- `outputFormat` (mandatory): the output format generated by the transformation
- `outputMimeType` (optional): the mime type for the generated output. In case it's missing, the `outputFormat` is assumed to be a mime type and used for the purpose.
- `fileExtension` (optional): the file extension for the generated output. In case it's missing `txt` will be used.
- `xslt` (mandatory): the name of XSLT 1.0 style sheet used for the transformation

The associated XSLT file will be `$GEOSERVER_DATA_DIR/wfs/transform/global.xslt` folder, and it will be able to transform any GML2 input into a corresponding HTML file. Here is an example:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:wfs="http://www.opengis.net/wfs"
  xmlns:tiger="http://www.census.gov" xmlns:gml="http://www.opengis.net/gml"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
      <body>
      <xsl:for-each select="wfs:FeatureCollection/gml:featureMember/*">
        <h2><xsl:value-of select="@fid"/></h2>
        <table border="1">
          <tr>
            <th>Attribute</th>
            <th>Value</th>
          </tr>
            <!-- [not(*)] strips away all nodes having
                 children, in particular, geometries -->
            <xsl:for-each select="./*[not(*)]">
            <tr>
              <td>
```

```
              <xsl:value-of select="name()" />
            </td>
            <td>
              <xsl:value-of select="." />
            </td>
          </tr>
          </xsl:for-each>
        </table>
      </xsl:for-each>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

15.18.3 Type specific transformations

Type specific transformations can refer to a specific type and leverage its attributes directly. While not required, it is good practice to setup a global transformation that can handle any feature type (since the output format is declared in the capabilities document as being general, not type specific) and then override it for specific feature types in order to create special transformations for them.

Here is an example of a transformation declaration that is type specific, that will be located at `$GEOSERVER_DATA_DIR/wfs/transform/html_bridges.xml`

```
<transform>
  <sourceFormat>text/xml; subtype=gml/2.1.2</sourceFormat>
  <outputFormat>HTML</outputFormat>
  <outputMimeType>text/html</outputMimeType>
  <fileExtension>html</fileExtension>
  <xslt>html_bridges.xslt</xslt>
  <featureType>
    <id>cite:Bridges</id>
  </featureType>
</transform>
```

The extra `featureType` element associates the transformation to the specific feature type

The associated xslt file will be located at `$GEOSERVER_DATA_DIR/wfs/transform/html_bridges.xslt` and will leveraging knowlegde about the input attributes:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:wfs="http://www.opengis.net/wfs"
  xmlns:cite="http://www.opengis.net/cite" xmlns:gml="http://www.opengis.net/gml"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
      <body>
        <h2>Bridges</h2>
        <xsl:for-each
            select="wfs:FeatureCollection/gml:featureMember/cite:Bridges">
        <ul>
          <li>ID: <xsl:value-of select="@fid" /></li>
          <li>FID: <xsl:value-of select="cite:FID" /></li>
          <li>Name: <xsl:value-of select="cite:NAME" /></li>
        </ul>
        <p/>
        </xsl:for-each>
```

```
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

Note: While writing the XSLT always remember to declare all prefixes used in the sheet in the `stylesheet` element, otherwise you might encounter hard to understand error messages

A specific feature type can be associated to multiple output formats. While uncommon, the same xslt file can also be associated to multiple feature types by creating multiple xml configuration files, and associating a different feature type in each.

15.18.4 Rest configuration

Transformations can be created, updated and deleted via the REST api (normally, this requires administrator privileges). Each transformation is represented with the same XML format used on disk, but with two variants:

- a new `name` attribute appears, which matches the XML file name
- the `featureType` element contains also a link to the resource representing the feature type in the REST config tree

For example:

```
<transform>
  <name>test</name>
  <sourceFormat>text/xml; subtype=gml/2.1.2</sourceFormat>
  <outputFormat>text/html</outputFormat>
  <fileExtension>html</fileExtension>
  <xslt>test-tx.xslt</xslt>
  <featureType>
    <name>tiger:poi</name>
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate" href="http://
↪localhost:8080/geoserver/rest/workspaces/cite/datastores/cite/featuretypes/bridges.
↪xml" type="application/xml"/>
  </featureType>
</transform>
```

Here is a list of resources and the HTTP methods that can be used on them.

`/rest/services/wfs/transforms[.<format>]`

Method	Action	Return Code	Formats	Default Format	Parameters
GET	List all available transforms	200	HTML, XML, JSON	HTML	
POST	Add a new transformation	201, with `Location` header	XML, JSON		name, sourceFormat, outputFormat, outputMimeType
PUT	Update global settings	200	XML, JSON		
DELETE		405			

The `POST` method can be used to create a transformation in two ways:

- if the content type used is `application/xml` the server will assume a `<transform>` definition is being posted, and the XSLT will have to be uploaded separately using a `PUT` request with content type `application/xslt+xml` against the transformation resource
- if the content type used is `application/xslt+xml` the server will assume the XSLT itself is being posted, and the `name`, `sourceFormat`, `outputFormat`, `outputMimeType` query parameters will be used to fill in the transform configuration instead

`/rest/services/wfs/transforms/<transform>[.<format>]`

Method	Action	Return Code	Formats	Default Format
GET	Returns the transformation	200	HTML, XML, XSLT	HTML
POST		405		
PUT	Updates either the transformation configuration, or its XSLT, depending on the mime type used	200	XML, XSLT	
DELETE	Deletes the transformation	200		

The PUT operation behaves differently depending on the content type used in the request:

- if the content type used is `application/xml` the server will assume a `<transform>` definition is being sent and will update it
- if the content type used is `application/xslt+xml` the server will assume the XSLT itself is being posted, as such the configuration won't be modified, but the XSLT associated to it will be overwritten instead

15.19 Web Coverage Service 2.0 Earth Observation extensions

The WCS 2.0 Earth Observation application profile (EO-WCS, OGC 10-140r1) extends the base WCS 2.0 protocol by adding temporal support and complex coverage structural description to the base WCS 2.0 protocol, in addition to requiring that a number of other extensions are supported - the base GeoServer WCS 2.0 module already supports all of those, such as subsetting and reprojection for example). The full specification can be downloaded from the OGC web site.

In the WCS 2.0 EO data model we not only have coverages, but also stitched mosaics (sets of coverages making up a mosaic of images, all granules having the same time and elevation) and dataset series, groups of coverages having different times and/or other attributes (elevation, custom dimensions). A dataset series is exposed in the capabilities document (inside the extension section) and its internal structure can be retrieved calling the new DescribeEOCoverageSet call. At the time of writing the EO extension adds support for dataset series, but does not provide direct support for stitched mosaic description.

Each grid layer exposing its inner structure will then expose a flag to enable its exposure as a dataset series. At the time of writing, the only grid readers capable of exposing their internal structure are image mosaic and netCDF.

15.19.1 Installing the WCS 2.0 EO extension

The steps to install the EO extension as the same as most other extensions:

- Go to the download page and look among the extensions

- Download the WCS 2.0 EO extension package (it's a zip file)
- Stop GeoServer (or the web container hosting it)
- Unpack the contents of the zip file in the geoserver/WEB-INF/lib folder
- Restart GeoServer

15.19.2 Exposing dataset series

The first step to work with EO is to go into the WCS service panel and enable the EO extensions:

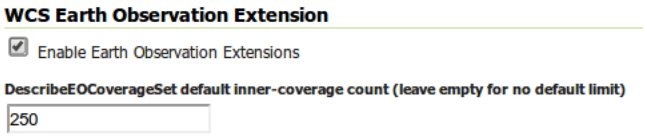


Fig. 15.43: *Enabling the WCS 2.0 EO extensions*

The second step is finding and activating the EO extensions for a suitable grid layer, which needs to be one with time dimension support and ability to describe its inner structure. At the time of writing, this means a image mosaic with time support or a netCDF data layer with time dimension. Once the layer is located, the EO extensions for it can be enabled by ticking a checkbox in the publishing tab:

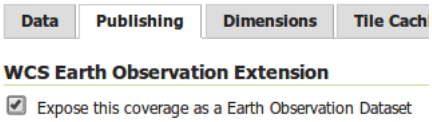


Fig. 15.44: *Exposing a layer as a dataset*

Once that is done the capabilities document (e.g. `http://localhost:8080/geoserver/ows?service=WCS&version=2.0.1&request=GetCapabilities` for WCS 2.0 will contain an indication that a coverage set is present:

```
<wcs:Extension>
  <wcseo:DatasetSeriesSummary>
    <ows:WGS84BoundingBox>
      <ows:LowerCorner>0.2372206885127698 40.562080748421806</ows:LowerCorner>
      <ows:UpperCorner>14.592757149389236 44.55808294568743</ows:UpperCorner>
    </ows:WGS84BoundingBox>
    <wcseo:DatasetSeriesId>nurc__watertemp_dss</wcseo:DatasetSeriesId>
    <gml:TimePeriod gml:id="nurc__watertemp_dss__timeperiod">
      <gml:beginPosition>2008-10-31T00:00:00.000Z</gml:beginPosition>
      <gml:endPosition>2008-11-01T00:00:00.000Z</gml:endPosition>
   </gml:TimePeriod>
  </wcseo:DatasetSeriesSummary>
</wcs:Extension>
```

And issuing a DescribeEOCoverageSet (e.g. `http://localhost:8080/geoserver/ows?service=WCS&version=2.0.1&request=DescribeEOCoverageSet&eoId=nurc__watertemp_dss`) on it will return the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<wcseo:EOCoverageSetDescription xmlns:eop="http://www.opengis.net/eop/2.0" xmlns:gml=
↪"http://www.opengis.net/gml/3.2" xmlns:wcsgs="http://www.geoserver.org/wcsgs/2.0"␣
↪xmlns:gmlcov="http://www.opengis.net/gmlcov/1.0" xmlns:om="http://www.opengis.net/
↪om/2.0" xmlns:swe="http://www.opengis.net/swe/2.0" xmlns:wcs="http://www.opengis.
↪net/wcs/2.0" xmlns:wcseo="http://www.opengis.net/wcseo/1.0" xmlns:xlink="http://www.
↪w3.org/1999/xlink" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"␣
↪numberMatched="4" numberReturned="4" xsi:schemaLocation="http://www.opengis.net/
↪wcseo/1.0 http://localhost:8080/geoserver/schemas/wcseo/1.0/wcsEOAll.xsd">
  <wcs:CoverageDescriptions>
    <wcs:CoverageDescription gml:id="nurc__watertemp_granule_watertemp.1">
      <gml:boundedBy>
        <gml:EnvelopeWithTimePeriod srsName="http://www.opengis.net/def/crs/EPSG/0/
↪4326" axisLabels="Lat Long time" uomLabels="Deg Deg s" srsDimension="2">
          <gml:lowerCorner>40.562080748421806 0.23722068851276978</gml:lowerCorner>
          <gml:upperCorner>44.55808294568743 14.592757149389236</gml:upperCorner>
          <gml:beginPosition>2008-11-01T00:00:00.000Z</gml:beginPosition>
          <gml:endPosition>2008-11-01T00:00:00.000Z</gml:endPosition>
        </gml:EnvelopeWithTimePeriod>
      </gml:boundedBy>
      <wcs:CoverageId>nurc__watertemp_granule_watertemp.1</wcs:CoverageId>
      <gml:coverageFunction>
        <gml:GridFunction>
          <gml:sequenceRule axisOrder="+2 +1">Linear</gml:sequenceRule>
          <gml:startPoint>0 0</gml:startPoint>
        </gml:GridFunction>
      </gml:coverageFunction>
      <gmlcov:metadata>
        <gmlcov:Extension>
          <wcsgs:TimeDomain default="2008-11-01T00:00:00.000Z">
            <gml:TimeInstant gml:id="nurc__watertemp_granule_watertemp.1_td_0">
              <gml:timePosition>2008-11-01T00:00:00.000Z</gml:timePosition>
            </gml:TimeInstant>
          </wcsgs:TimeDomain>
          <wcseo:EOMetadata>
            <eop:EarthObservation gml:id="nurc__watertemp_metadata">
              <om:phenomenonTime>
                <gml:TimePeriod gml:id="nurc__watertemp_tp">
                  <gml:beginPosition>2008-11-01T00:00:00.000Z</gml:beginPosition>
                  <gml:endPosition>2008-11-01T00:00:00.000Z</gml:endPosition>
                </gml:TimePeriod>
              </om:phenomenonTime>
              <om:resultTime>
                <gml:TimeInstant gml:id="nurc__watertemp_rt">
                  <gml:timePosition>2008-11-01T00:00:00.000Z</gml:timePosition>
                </gml:TimeInstant>
              </om:resultTime>
              <om:procedure/>
              <om:observedProperty/>
              <om:FeatureOfInterest>
                <eop:Footprint gml:id="nurc__watertemp_fp">
                  <eop:multiExtentOf>
                    <gml:MultiSurface gml:id="nurc__watertemp_ms" srsName="http://www.
↪opengis.net/def/crs/EPSG/0/4326">
                      <gml:surfaceMembers>
                        <gml:Polygon gml:id="nurc__watertemp_msp">
                          <gml:exterior>
                            <gml:LinearRing>
```

```
                              <gml:posList>40.562080748421806 0.23722068851276978 40.
↪562080748421806 14.592757149389236 44.55808294568743 14.592757149389236 44.
↪55808294568743 0.23722068851276978 40.562080748421806 0.23722068851276978</
↪gml:posList>
                            </gml:LinearRing>
                          </gml:exterior>
                        </gml:Polygon>
                      </gml:surfaceMembers>
                    </gml:MultiSurface>
                  </eop:multiExtentOf>
                  <eop:centerOf>
                    <gml:Point gml:id="nurc__watertemp_co" srsName="http://www.
↪opengis.net/def/crs/EPSG/0/4326">
                      <gml:pos>42.56008184705462 7.4149889189510025</gml:pos>
                    </gml:Point>
                  </eop:centerOf>
                </eop:Footprint>
              </om:FeatureOfInterest>
              <eop:metaDataProperty>
                <eop:EarthObservationMetaData>
                  <eop:identifier>nurc__watertemp</eop:identifier>
                  <eop:acquisitionType>NOMINAL</eop:acquisitionType>
                  <eop:status>ARCHIVED</eop:status>
                </eop:EarthObservationMetaData>
              </eop:metaDataProperty>
            </eop:EarthObservation>
          </wcseo:EOMetadata>
        </gmlcov:Extension>
      </gmlcov:metadata>
      <gml:domainSet>
        <gml:RectifiedGrid gml:id="grid00__nurc__watertemp_granule_watertemp.1"␣
↪dimension="2">
          <gml:limits>
            <gml:GridEnvelope>
              <gml:low>0 0</gml:low>
              <gml:high>24 24</gml:high>
            </gml:GridEnvelope>
          </gml:limits>
          <gml:axisLabels>i j</gml:axisLabels>
          <gml:origin>
            <gml:Point gml:id="p00_nurc__watertemp_granule_watertemp.1" srsName=
↪"http://www.opengis.net/def/crs/EPSG/0/4326">
              <gml:pos>44.47816290174212 0.5243314177302991</gml:pos>
            </gml:Point>
          </gml:origin>
          <gml:offsetVector srsName="http://www.opengis.net/def/crs/EPSG/0/4326">0.0␣
↪0.5742214584350587</gml:offsetVector>
          <gml:offsetVector srsName="http://www.opengis.net/def/crs/EPSG/0/4326">-0.
↪159840087890625 0.0</gml:offsetVector>
        </gml:RectifiedGrid>
      </gml:domainSet>
      <gmlcov:rangeType>
        <swe:DataRecord>
          <swe:field name="GRAY_INDEX">
            <swe:Quantity>
              <swe:description>GRAY_INDEX</swe:description>
              <swe:uom code="W.m-2.Sr-1"/>
              <swe:constraint>
```

```
                <swe:AllowedValues>
                  <swe:interval>-1.7976931348623157E308 1.7976931348623157E308</
↪swe:interval>
                </swe:AllowedValues>
              </swe:constraint>
            </swe:Quantity>
          </swe:field>
        </swe:DataRecord>
      </gmlcov:rangeType>
      <wcs:ServiceParameters>
        <wcs:CoverageSubtype>RectifiedGridCoverage</wcs:CoverageSubtype>
        <wcs:nativeFormat>image/tiff</wcs:nativeFormat>
      </wcs:ServiceParameters>
    </wcs:CoverageDescription>
    <wcs:CoverageDescription gml:id="nurc__watertemp_granule_watertemp.2">
      <gml:boundedBy>
        <gml:EnvelopeWithTimePeriod srsName="http://www.opengis.net/def/crs/EPSG/0/
↪4326" axisLabels="Lat Long time" uomLabels="Deg Deg s" srsDimension="2">
          <gml:lowerCorner>40.562080748421806 0.23722068851276978</gml:lowerCorner>
          <gml:upperCorner>44.55808294568743 14.592757149389236</gml:upperCorner>
          <gml:beginPosition>2008-11-01T00:00:00.000Z</gml:beginPosition>
          <gml:endPosition>2008-11-01T00:00:00.000Z</gml:endPosition>
        </gml:EnvelopeWithTimePeriod>
      </gml:boundedBy>
      <wcs:CoverageId>nurc__watertemp_granule_watertemp.2</wcs:CoverageId>
      <gml:coverageFunction>
        <gml:GridFunction>
          <gml:sequenceRule axisOrder="+2 +1">Linear</gml:sequenceRule>
          <gml:startPoint>0 0</gml:startPoint>
        </gml:GridFunction>
      </gml:coverageFunction>
      <gmlcov:metadata>
        <gmlcov:Extension>
          <wcsgs:TimeDomain default="2008-11-01T00:00:00.000Z">
            <gml:TimeInstant gml:id="nurc__watertemp_granule_watertemp.2_td_0">
              <gml:timePosition>2008-11-01T00:00:00.000Z</gml:timePosition>
            </gml:TimeInstant>
          </wcsgs:TimeDomain>
          <wcseo:EOMetadata>
            <eop:EarthObservation gml:id="nurc__watertemp_metadata">
              <om:phenomenonTime>
                <gml:TimePeriod gml:id="nurc__watertemp_tp">
                  <gml:beginPosition>2008-11-01T00:00:00.000Z</gml:beginPosition>
                  <gml:endPosition>2008-11-01T00:00:00.000Z</gml:endPosition>
                </gml:TimePeriod>
              </om:phenomenonTime>
              <om:resultTime>
                <gml:TimeInstant gml:id="nurc__watertemp_rt">
                  <gml:timePosition>2008-11-01T00:00:00.000Z</gml:timePosition>
                </gml:TimeInstant>
              </om:resultTime>
              <om:procedure/>
              <om:observedProperty/>
              <om:FeatureOfInterest>
                <eop:Footprint gml:id="nurc__watertemp_fp">
                  <eop:multiExtentOf>
                    <gml:MultiSurface gml:id="nurc__watertemp_ms" srsName="http://www.
↪opengis.net/def/crs/EPSG/0/4326">
```

```
                    <gml:surfaceMembers>
                      <gml:Polygon gml:id="nurc__watertemp_msp">
                        <gml:exterior>
                          <gml:LinearRing>
                            <gml:posList>40.562080748421806 0.23722068851276978 40.
↪562080748421806 14.592757149389236 44.55808294568743 14.592757149389236 44.
↪55808294568743 0.23722068851276978 40.562080748421806 0.23722068851276978</
↪gml:posList>
                          </gml:LinearRing>
                        </gml:exterior>
                      </gml:Polygon>
                    </gml:surfaceMembers>
                  </gml:MultiSurface>
                </eop:multiExtentOf>
                <eop:centerOf>
                  <gml:Point gml:id="nurc__watertemp_co" srsName="http://www.
↪opengis.net/def/crs/EPSG/0/4326">
                    <gml:pos>42.56008184705462 7.4149889189510025</gml:pos>
                  </gml:Point>
                </eop:centerOf>
              </eop:Footprint>
            </om:FeatureOfInterest>
            <eop:metaDataProperty>
              <eop:EarthObservationMetaData>
                <eop:identifier>nurc__watertemp</eop:identifier>
                <eop:acquisitionType>NOMINAL</eop:acquisitionType>
                <eop:status>ARCHIVED</eop:status>
              </eop:EarthObservationMetaData>
            </eop:metaDataProperty>
          </eop:EarthObservation>
        </wcseo:EOMetadata>
      </gmlcov:Extension>
    </gmlcov:metadata>
    <gml:domainSet>
      <gml:RectifiedGrid gml:id="grid00__nurc__watertemp_granule_watertemp.2"␣
↪dimension="2">
        <gml:limits>
          <gml:GridEnvelope>
            <gml:low>0 0</gml:low>
            <gml:high>24 24</gml:high>
          </gml:GridEnvelope>
        </gml:limits>
        <gml:axisLabels>i j</gml:axisLabels>
        <gml:origin>
          <gml:Point gml:id="p00_nurc__watertemp_granule_watertemp.2" srsName=
↪"http://www.opengis.net/def/crs/EPSG/0/4326">
            <gml:pos>44.47816290174212 0.5243314177302991</gml:pos>
          </gml:Point>
        </gml:origin>
        <gml:offsetVector srsName="http://www.opengis.net/def/crs/EPSG/0/4326">0.0␣
↪0.5742214584350587</gml:offsetVector>
        <gml:offsetVector srsName="http://www.opengis.net/def/crs/EPSG/0/4326">-0.
↪159840087890625 0.0</gml:offsetVector>
      </gml:RectifiedGrid>
    </gml:domainSet>
    <gmlcov:rangeType>
      <swe:DataRecord>
        <swe:field name="GRAY_INDEX">
```

```
            <swe:Quantity>
              <swe:description>GRAY_INDEX</swe:description>
              <swe:uom code="W.m-2.Sr-1"/>
              <swe:constraint>
                <swe:AllowedValues>
                  <swe:interval>-1.7976931348623157E308 1.7976931348623157E308</
↪swe:interval>
                </swe:AllowedValues>
              </swe:constraint>
            </swe:Quantity>
          </swe:field>
        </swe:DataRecord>
      </gmlcov:rangeType>
      <wcs:ServiceParameters>
        <wcs:CoverageSubtype>RectifiedGridCoverage</wcs:CoverageSubtype>
        <wcs:nativeFormat>image/tiff</wcs:nativeFormat>
      </wcs:ServiceParameters>
    </wcs:CoverageDescription>
    <wcs:CoverageDescription gml:id="nurc__watertemp_granule_watertemp.3">
      <gml:boundedBy>
        <gml:EnvelopeWithTimePeriod srsName="http://www.opengis.net/def/crs/EPSG/0/
↪4326" axisLabels="Lat Long time" uomLabels="Deg Deg s" srsDimension="2">
          <gml:lowerCorner>40.562080748421806 0.23722068851276978</gml:lowerCorner>
          <gml:upperCorner>44.55808294568743 14.592757149389236</gml:upperCorner>
          <gml:beginPosition>2008-10-31T00:00:00.000Z</gml:beginPosition>
          <gml:endPosition>2008-10-31T00:00:00.000Z</gml:endPosition>
        </gml:EnvelopeWithTimePeriod>
      </gml:boundedBy>
      <wcs:CoverageId>nurc__watertemp_granule_watertemp.3</wcs:CoverageId>
      <gml:coverageFunction>
        <gml:GridFunction>
          <gml:sequenceRule axisOrder="+2 +1">Linear</gml:sequenceRule>
          <gml:startPoint>0 0</gml:startPoint>
        </gml:GridFunction>
      </gml:coverageFunction>
      <gmlcov:metadata>
        <gmlcov:Extension>
          <wcsgs:TimeDomain default="2008-10-31T00:00:00.000Z">
            <gml:TimeInstant gml:id="nurc__watertemp_granule_watertemp.3_td_0">
              <gml:timePosition>2008-10-31T00:00:00.000Z</gml:timePosition>
            </gml:TimeInstant>
          </wcsgs:TimeDomain>
          <wcseo:EOMetadata>
            <eop:EarthObservation gml:id="nurc__watertemp_metadata">
              <om:phenomenonTime>
                <gml:TimePeriod gml:id="nurc__watertemp_tp">
                  <gml:beginPosition>2008-10-31T00:00:00.000Z</gml:beginPosition>
                  <gml:endPosition>2008-10-31T00:00:00.000Z</gml:endPosition>
                </gml:TimePeriod>
              </om:phenomenonTime>
              <om:resultTime>
                <gml:TimeInstant gml:id="nurc__watertemp_rt">
                  <gml:timePosition>2008-10-31T00:00:00.000Z</gml:timePosition>
                </gml:TimeInstant>
              </om:resultTime>
              <om:procedure/>
              <om:observedProperty/>
              <om:FeatureOfInterest>
```

```
                <eop:Footprint gml:id="nurc__watertemp_fp">
                  <eop:multiExtentOf>
                    <gml:MultiSurface gml:id="nurc__watertemp_ms" srsName="http://www.
↪opengis.net/def/crs/EPSG/0/4326">
                      <gml:surfaceMembers>
                        <gml:Polygon gml:id="nurc__watertemp_msp">
                          <gml:exterior>
                            <gml:LinearRing>
                              <gml:posList>40.562080748421806 0.23722068851276978 40.
↪562080748421806 14.592757149389236 44.55808294568743 14.592757149389236 44.
↪55808294568743 0.23722068851276978 40.562080748421806 0.23722068851276978</
↪gml:posList>
                            </gml:LinearRing>
                          </gml:exterior>
                        </gml:Polygon>
                      </gml:surfaceMembers>
                    </gml:MultiSurface>
                  </eop:multiExtentOf>
                  <eop:centerOf>
                    <gml:Point gml:id="nurc__watertemp_co" srsName="http://www.
↪opengis.net/def/crs/EPSG/0/4326">
                      <gml:pos>42.56008184705462 7.4149889189510025</gml:pos>
                    </gml:Point>
                  </eop:centerOf>
                </eop:Footprint>
              </om:FeatureOfInterest>
              <eop:metaDataProperty>
                <eop:EarthObservationMetaData>
                  <eop:identifier>nurc__watertemp</eop:identifier>
                  <eop:acquisitionType>NOMINAL</eop:acquisitionType>
                  <eop:status>ARCHIVED</eop:status>
                </eop:EarthObservationMetaData>
              </eop:metaDataProperty>
            </eop:EarthObservation>
          </wcseo:EOMetadata>
        </gmlcov:Extension>
      </gmlcov:metadata>
      <gml:domainSet>
        <gml:RectifiedGrid gml:id="grid00__nurc__watertemp_granule_watertemp.3"␣
↪dimension="2">
          <gml:limits>
            <gml:GridEnvelope>
              <gml:low>0 0</gml:low>
              <gml:high>24 24</gml:high>
            </gml:GridEnvelope>
          </gml:limits>
          <gml:axisLabels>i j</gml:axisLabels>
          <gml:origin>
            <gml:Point gml:id="p00_nurc__watertemp_granule_watertemp.3" srsName=
↪"http://www.opengis.net/def/crs/EPSG/0/4326">
              <gml:pos>44.47816290174212 0.5243314177302991</gml:pos>
            </gml:Point>
          </gml:origin>
          <gml:offsetVector srsName="http://www.opengis.net/def/crs/EPSG/0/4326">0.0␣
↪0.5742214584350587</gml:offsetVector>
          <gml:offsetVector srsName="http://www.opengis.net/def/crs/EPSG/0/4326">-0.
↪159840087890625 0.0</gml:offsetVector>
        </gml:RectifiedGrid>
```

```
      </gml:domainSet>
      <gmlcov:rangeType>
        <swe:DataRecord>
          <swe:field name="GRAY_INDEX">
            <swe:Quantity>
              <swe:description>GRAY_INDEX</swe:description>
              <swe:uom code="W.m-2.Sr-1"/>
              <swe:constraint>
                <swe:AllowedValues>
                  <swe:interval>-1.7976931348623157E308 1.7976931348623157E308</
↪swe:interval>
                </swe:AllowedValues>
              </swe:constraint>
            </swe:Quantity>
          </swe:field>
        </swe:DataRecord>
      </gmlcov:rangeType>
      <wcs:ServiceParameters>
        <wcs:CoverageSubtype>RectifiedGridCoverage</wcs:CoverageSubtype>
        <wcs:nativeFormat>image/tiff</wcs:nativeFormat>
      </wcs:ServiceParameters>
    </wcs:CoverageDescription>
    <wcs:CoverageDescription gml:id="nurc__watertemp_granule_watertemp.4">
      <gml:boundedBy>
        <gml:EnvelopeWithTimePeriod srsName="http://www.opengis.net/def/crs/EPSG/0/
↪4326" axisLabels="Lat Long time" uomLabels="Deg Deg s" srsDimension="2">
          <gml:lowerCorner>40.562080748421806 0.23722068851276978</gml:lowerCorner>
          <gml:upperCorner>44.55808294568743 14.592757149389236</gml:upperCorner>
          <gml:beginPosition>2008-10-31T00:00:00.000Z</gml:beginPosition>
          <gml:endPosition>2008-10-31T00:00:00.000Z</gml:endPosition>
        </gml:EnvelopeWithTimePeriod>
      </gml:boundedBy>
      <wcs:CoverageId>nurc__watertemp_granule_watertemp.4</wcs:CoverageId>
      <gml:coverageFunction>
        <gml:GridFunction>
          <gml:sequenceRule axisOrder="+2 +1">Linear</gml:sequenceRule>
          <gml:startPoint>0 0</gml:startPoint>
        </gml:GridFunction>
      </gml:coverageFunction>
      <gmlcov:metadata>
        <gmlcov:Extension>
          <wcsgs:TimeDomain default="2008-10-31T00:00:00.000Z">
            <gml:TimeInstant gml:id="nurc__watertemp_granule_watertemp.4_td_0">
              <gml:timePosition>2008-10-31T00:00:00.000Z</gml:timePosition>
            </gml:TimeInstant>
          </wcsgs:TimeDomain>
          <wcseo:EOMetadata>
            <eop:EarthObservation gml:id="nurc__watertemp_metadata">
              <om:phenomenonTime>
                <gml:TimePeriod gml:id="nurc__watertemp_tp">
                  <gml:beginPosition>2008-10-31T00:00:00.000Z</gml:beginPosition>
                  <gml:endPosition>2008-10-31T00:00:00.000Z</gml:endPosition>
                </gml:TimePeriod>
              </om:phenomenonTime>
              <om:resultTime>
                <gml:TimeInstant gml:id="nurc__watertemp_rt">
                  <gml:timePosition>2008-10-31T00:00:00.000Z</gml:timePosition>
                </gml:TimeInstant>
```

```
            </om:resultTime>
            <om:procedure/>
            <om:observedProperty/>
            <om:FeatureOfInterest>
              <eop:Footprint gml:id="nurc__watertemp_fp">
                <eop:multiExtentOf>
                  <gml:MultiSurface gml:id="nurc__watertemp_ms" srsName="http://www.
↪opengis.net/def/crs/EPSG/0/4326">
                    <gml:surfaceMembers>
                      <gml:Polygon gml:id="nurc__watertemp_msp">
                        <gml:exterior>
                          <gml:LinearRing>
                            <gml:posList>40.562080748421806 0.23722068851276978 40.
↪562080748421806 14.592757149389236 44.55808294568743 14.592757149389236 44.
↪55808294568743 0.23722068851276978 40.562080748421806 0.23722068851276978</
↪gml:posList>
                          </gml:LinearRing>
                        </gml:exterior>
                      </gml:Polygon>
                    </gml:surfaceMembers>
                  </gml:MultiSurface>
                </eop:multiExtentOf>
                <eop:centerOf>
                  <gml:Point gml:id="nurc__watertemp_co" srsName="http://www.
↪opengis.net/def/crs/EPSG/0/4326">
                    <gml:pos>42.56008184705462 7.4149889189510025</gml:pos>
                  </gml:Point>
                </eop:centerOf>
              </eop:Footprint>
            </om:FeatureOfInterest>
            <eop:metaDataProperty>
              <eop:EarthObservationMetaData>
                <eop:identifier>nurc__watertemp</eop:identifier>
                <eop:acquisitionType>NOMINAL</eop:acquisitionType>
                <eop:status>ARCHIVED</eop:status>
              </eop:EarthObservationMetaData>
            </eop:metaDataProperty>
          </eop:EarthObservation>
        </wcseo:EOMetadata>
      </gmlcov:Extension>
    </gmlcov:metadata>
    <gml:domainSet>
      <gml:RectifiedGrid gml:id="grid00__nurc__watertemp_granule_watertemp.4"␣
↪dimension="2">
        <gml:limits>
          <gml:GridEnvelope>
            <gml:low>0 0</gml:low>
            <gml:high>24 24</gml:high>
          </gml:GridEnvelope>
        </gml:limits>
        <gml:axisLabels>i j</gml:axisLabels>
        <gml:origin>
          <gml:Point gml:id="p00_nurc__watertemp_granule_watertemp.4" srsName=
↪"http://www.opengis.net/def/crs/EPSG/0/4326">
            <gml:pos>44.47816290174212 0.5243314177302991</gml:pos>
          </gml:Point>
        </gml:origin>
        <gml:offsetVector srsName="http://www.opengis.net/def/crs/EPSG/0/4326">0.0␣
↪0.5742214584350587</gml:offsetVector>
```

```
          <gml:offsetVector srsName="http://www.opengis.net/def/crs/EPSG/0/4326">-0.
↪159840087890625 0.0</gml:offsetVector>
        </gml:RectifiedGrid>
      </gml:domainSet>
      <gmlcov:rangeType>
        <swe:DataRecord>
          <swe:field name="GRAY_INDEX">
            <swe:Quantity>
              <swe:description>GRAY_INDEX</swe:description>
              <swe:uom code="W.m-2.Sr-1"/>
              <swe:constraint>
                <swe:AllowedValues>
                  <swe:interval>-1.7976931348623157E308 1.7976931348623157E308</
↪swe:interval>
                </swe:AllowedValues>
              </swe:constraint>
            </swe:Quantity>
          </swe:field>
        </swe:DataRecord>
      </gmlcov:rangeType>
      <wcs:ServiceParameters>
        <wcs:CoverageSubtype>RectifiedGridCoverage</wcs:CoverageSubtype>
        <wcs:nativeFormat>image/tiff</wcs:nativeFormat>
      </wcs:ServiceParameters>
    </wcs:CoverageDescription>
  </wcs:CoverageDescriptions>
  <wcseo:DatasetSeriesDescriptions>
    <wcseo:DatasetSeriesDescription gml:id="nurc__watertemp_dss">
      <gml:boundedBy>
        <gml:Envelope srsName="http://www.opengis.net/def/crs/EPSG/0/4326" axisLabels=
↪"Lat Long" uomLabels="Deg Deg" srsDimension="2">
          <gml:lowerCorner>40.562080748421806 0.23722068851276978</gml:lowerCorner>
          <gml:upperCorner>44.55808294568743 14.592757149389236</gml:upperCorner>
        </gml:Envelope>
      </gml:boundedBy>
      <wcseo:DatasetSeriesId>nurc__watertemp_dss</wcseo:DatasetSeriesId>
      <gml:TimePeriod gml:id="nurc__watertemp_dss_timeperiod">
        <gml:beginPosition>2008-10-31T00:00:00.000Z</gml:beginPosition>
        <gml:endPosition>2008-11-01T00:00:00.000Z</gml:endPosition>
      </gml:TimePeriod>
    </wcseo:DatasetSeriesDescription>
  </wcseo:DatasetSeriesDescriptions>
</wcseo:EOCoverageSetDescription>
```

Any of the inner coverages can be then retrieved via a standard GetCoverage, even if it's not directly part of the capabilities document, for example, to retrieve the first granule in the watertemp layer the request would be:

```
http://localhost:8080/geoserver/ows?service=WCS&version=2.0.1&request=GetCoverage&
↪coverageId=nurc__watertemp_granule_watertemp.1
```

15.20 MongoDB Data Store

This module provides support for MongoDB data store. This extension is build on top of GeoTools MongoDB plugin.

15.20.1 Installation

1. Navigate to the GeoServer download page.
2. Find the page that matches the version of the running GeoServer.
3. Download the MongoDB extension. The download link will be in the *Extensions* section under *Vector Formats*.
4. Extract the files in this archive to the `WEB-INF/lib` directory of your GeoServer installation.
5. Restart GeoServer

15.20.2 Usage

If the extension was successfully installed a new type of data store named `MongoDB` should be available:

Fig. 15.45: *MongoDB data store.*

Configuring a new MongoDB data store requires providing:

1. The URL of a MongoDB database.
2. The absolute path to a data directory where GeoServer will store the schema produced for the published collections.

For more details about the usage of this data store please check the GeoTools MongoDB plugin documentation.

15.21 SLD REST Service

The SLD Service is a GeoServer REST service that can be used to create SLD styles on published GeoServer layers doing a classification on the layer data, following user provided directives.

The purpose of the service is to allow clients to dinamically publish data and create simple styles on it.

All the services are published under the common prefix `/rest/sldservice/{layer}`, where **layer** is the layer to classify/query.

15.21.1 Query Vector Data Attributes

`/attributes[.<format>]`

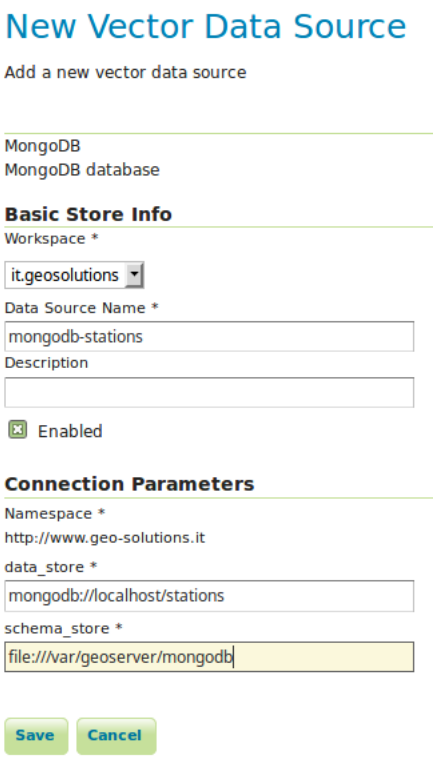


Fig. 15.46: *Configuring a MongoDB data store.*

Method	Action	Status code	Formats	Default Format
GET	Gets the list of attributes for the given layer (of vector type)	200	HTML, XML, JSON	HTML

The service can be used to get the attributes list for the given vector layer. This can be used by a client as a prerequisite for the **classify** service, to get all the attributes usable for classification and let the user choose one.

Examples

Get attributes for the states layer, in XML format

```
curl -v -u admin:geoserver -XGET
  http://localhost:8080/geoserver/rest/sldservice/states/attributes.xml
```

```
<Attributes layer="states">
  <Attribute>
    <name>P_FEMALE</name>
    <type>Double</type>
  </Attribute>
  <Attribute>
    <name>HOUSHOLD</name>
    <type>Double</type>
  </Attribute>
```

```
  <Attribute>
    <name>SERVICE</name>
    <type>Double</type>
  </Attribute>
  ...
</Attributes>
```

Get attributes for the states layer, in JSON format

```
curl -v -u admin:geoserver -XGET
  http://localhost:8080/geoserver/rest/sldservice/states/attributes.json
```

```
{
   "Attributes":{
      "@layer":"states",
      "Attribute":[
         {
            "name":"P_FEMALE",
            "type":"Double"
         },
         {
            "name":"HOUSHOLD",
            "type":"Double"
         },
         {
            "name":"SERVICE",
            "type":"Double"
         },
         ...
      ]
   }
}
```

15.21.2 Classify Raster and Vector Data

`/classify[.<format>]`

Method	Action	Status code	Formats	Default Format
GET	Create a set of SLD Rules for the given layer	200	HTML, XML, JSON	HTML

The service can be used to create a set of SLD rules for the given vector layer, specifying the **attribute** used for classification, the **classification type** (equalInterval, uniqueInterval, quantile, jenks, equalArea) and one of the **predefined color ranges** (red, blue, gray, jet, random, custom), together with some other optional parameters.

The same can be applied on a raster layer too, in order to classify its contents. Data from the first band is used by default, but a different one can be selected.

Using the **CUSTOM** ColorMap, startColor and endColor (and optionally midColor) have to be specified.

The parameters usable to customize the ColorMap are:

Parameter	Description	Values	Default Value
intervals	Number of intervals (rules) for the SLD	integer numeric value	2
attribute (mandatory)	Classification attribute	For vector layers, one of the layer attribute names, for raster layers, a band number (starting from one, like in the raster symbolizer)	No default for vectors, "1" for rasters
method	Classification method	equalInterval, uniqueInterval, quantile, jenks, equalArea	equalInterval
open	open or closed ranges	true, false	false
reverse	normal or inverted ranges	true, false	false
normalize	normalize (cast) attribute to double type (needed by some stores to handle integer types correctly)	true, false	false
ramp	color ranges to use	red, blue, gray, jet, random, custom	red
startColor	starting color for the custom ramp		
endColor	ending color for the custom ramp		
midColor	central color for the custom ramp		
colors	list of comma delimited colors for the custom ramp (use this instead of startColor, endColor and midColor to specify colors in more detail)		
strokeColor	color of the stroke, for points and polygons		BLACK
strokeWeight	weight of the stroke, for points and polygons (use a negative value to not include stroke in style)		1
pointSize	size of points		15
fullSLD	create a full valid SLD document, instead of the Rules fragment only	true or false	false
cache	append caching headers to the responses	expire time in seconds, use 0 to disable cache	600 (10 minutes)
viewparams	allows use of parametric views	view parameters in the usual format (<key>:<value>;...;<keyN>:<valueN>)	
customClasses	allows specifying a set of custom classes (client driven style); no classes calculation will happen (method, intervals, etc. are ignored)	classes in the following format: <min>,<max>,<color>;...;<minN>,<maxN>,<colorN>	
bbox	allows to run the classification on a specific bounding box. Recommended when the overall dataset is too big, and the classification can be performed on a smaller dataset, or to enhance the visualization of a particular subset of data	same syntax as WMS/WFS, expected axis order is east/north unless the spatial reference system is explicitly provided, `minx,miny,max,maxy[,srsName]`	
stddevs	limits the data the classifier is working on to a range of "stddevs" standard deviations around the mean value.	a positive floating point number (e.g., '1', '2.5', '3').	
env	a list of environment variables that the underlying layer may be using to select features/rasters to be classified (e.g., by using the `filter` in vec-	a semicolon separate list of name to value assignments, e.g., `name1:value1;`	

Examples

A default (equalInterval) classification on the states layer LAND_KM attribute using a red based color range.

```
curl -v -u admin:geoserver -XGET
  http://localhost:8080/geoserver/rest/sldservice/states/classify.xml?attribute=LAND_
↪KM&ramp=red
```

```
<Rules>
  <Rule>
    <Title> &gt; 159.1 AND &lt;= 344189.1</Title>
    <Filter>
      <And>
        <PropertyIsGreaterThanOrEqualTo>
          <PropertyName>LAND_KM</PropertyName>
          <Literal>159.1</Literal>
        </PropertyIsGreaterThanOrEqualTo>
        <PropertyIsLessThanOrEqualTo>
          <PropertyName>LAND_KM</PropertyName>
          <Literal>344189.1</Literal>
        </PropertyIsLessThanOrEqualTo>
      </And>
    </Filter>
    <PolygonSymbolizer>
      <Fill>
        <CssParameter name="fill">#680000</CssParameter>
      </Fill>
      <Stroke/>
    </PolygonSymbolizer>
  </Rule>
  <Rule>
    <Title> &gt; 344189.1 AND &lt;= 688219.2</Title>
    <Filter>
      <And>
        <PropertyIsGreaterThan>
          <PropertyName>LAND_KM</PropertyName>
          <Literal>344189.1</Literal>
        </PropertyIsGreaterThan>
        <PropertyIsLessThanOrEqualTo>
          <PropertyName>LAND_KM</PropertyName>
          <Literal>688219.2</Literal>
        </PropertyIsLessThanOrEqualTo>
      </And>
    </Filter>
    <PolygonSymbolizer>
      <Fill>
        <CssParameter name="fill">#B20000</CssParameter>
      </Fill>
      <Stroke/>
    </PolygonSymbolizer>
  </Rule>
</Rules>
```

A uniqueInterval classification on the states layer SUB_REGION attribute using a red based color range.

```
curl -v -u admin:geoserver -XGET
  http://localhost:8080/geoserver/rest/sldservice/states/classify.xml?attribute=SUB_
↪REGION&ramp=red&method=uniqueInterval
```

```
<Rules>
  <Rule>
    <Title>E N Cen</Title>
    <Filter>
      <PropertyIsEqualTo>
        <PropertyName>SUB_REGION</PropertyName>
        <Literal>E N Cen</Literal>
      </PropertyIsEqualTo>
    </Filter>
    <PolygonSymbolizer>
      <Fill>
        <CssParameter name="fill">#330000</CssParameter>
      </Fill>
      <Stroke/>
    </PolygonSymbolizer>
  </Rule>
  <Rule>
    <Title>E S Cen</Title>
    <Filter>
      <PropertyIsEqualTo>
        <PropertyName>SUB_REGION</PropertyName>
        <Literal>E S Cen</Literal>
      </PropertyIsEqualTo>
    </Filter>
    <PolygonSymbolizer>
      <Fill>
        <CssParameter name="fill">#490000</CssParameter>
      </Fill>
      <Stroke/>
    </PolygonSymbolizer>
  </Rule>
  ...
</Rules>
```

A uniqueInterval classification on the states layer SUB_REGION attribute using a red based color range and 3 intervals.

```
curl -v -u admin:geoserver -XGET
  http://localhost:8080/geoserver/rest/sldservice/states/classify.xml?attribute=SUB_
↪REGION&ramp=red&method=uniqueInterval&intervals=3
```

```
<string>Intervals: 9</string>
```

A quantile classification on the states layer PERSONS attribute with a custom color ramp and 3 **closed** intervals.

```
curl -v -u admin:geoserver -XGET
  http://localhost:8080/geoserver/rest/sldservice/states/classify.xml?
↪attribute=PERSONS&ramp=CUSTOM&method=quantile&intervals=3&startColor=0xFF0000&
↪endColor=0x0000FF
```

```
<Rules>
  <Rule>
    <Title> &gt; 453588.0 AND &lt;= 2477574.0</Title>
    <Filter>
      <And>
```

```
      <PropertyIsGreaterThanOrEqualTo>
        <PropertyName>PERSONS</PropertyName>
        <Literal>453588.0</Literal>
      </PropertyIsGreaterThanOrEqualTo>
      <PropertyIsLessThanOrEqualTo>
        <PropertyName>PERSONS</PropertyName>
        <Literal>2477574.0</Literal>
      </PropertyIsLessThanOrEqualTo>
    </And>
  </Filter>
  <PolygonSymbolizer>
    <Fill>
      <CssParameter name="fill">#FF0000</CssParameter>
    </Fill>
    <Stroke/>
  </PolygonSymbolizer>
</Rule>
<Rule>
  <Title> &gt; 2477574.0 AND &lt;= 4866692.0</Title>
  <Filter>
    <And>
      <PropertyIsGreaterThan>
        <PropertyName>PERSONS</PropertyName>
        <Literal>2477574.0</Literal>
      </PropertyIsGreaterThan>
      <PropertyIsLessThanOrEqualTo>
        <PropertyName>PERSONS</PropertyName>
        <Literal>4866692.0</Literal>
      </PropertyIsLessThanOrEqualTo>
    </And>
  </Filter>
  <PolygonSymbolizer>
    <Fill>
      <CssParameter name="fill">#AA0055</CssParameter>
    </Fill>
    <Stroke/>
  </PolygonSymbolizer>
</Rule>
<Rule>
  <Title> &gt; 4866692.0 AND &lt;= 2.9760021E7</Title>
  <Filter>
    <And>
      <PropertyIsGreaterThan>
        <PropertyName>PERSONS</PropertyName>
        <Literal>4866692.0</Literal>
      </PropertyIsGreaterThan>
      <PropertyIsLessThanOrEqualTo>
        <PropertyName>PERSONS</PropertyName>
        <Literal>2.9760021E7</Literal>
      </PropertyIsLessThanOrEqualTo>
    </And>
  </Filter>
  <PolygonSymbolizer>
    <Fill>
      <CssParameter name="fill">#5500AA</CssParameter>
    </Fill>
    <Stroke/>
  </PolygonSymbolizer>
```

```
  </Rule>
</Rules>
```

A quantile classification on the states layer PERSONS attribute with a custom color ramp and 3 **open** intervals.

```
curl -v -u admin:geoserver -XGET
  http://localhost:8080/geoserver/rest/sldservice/states/classify.xml?
↪attribute=PERSONS&ramp=CUSTOM&method=quantile&intervals=3&startColor=0xFF0000&
↪endColor=0x0000FF&open=true
```

```
<Rules>
  <Rule>
    <Title> &lt;= 2477574.0</Title>
    <Filter>
      <PropertyIsLessThanOrEqualTo>
        <PropertyName>PERSONS</PropertyName>
        <Literal>2477574.0</Literal>
      </PropertyIsLessThanOrEqualTo>
    </Filter>
    <PolygonSymbolizer>
      <Fill>
        <CssParameter name="fill">#FF0000</CssParameter>
      </Fill>
      <Stroke/>
    </PolygonSymbolizer>
  </Rule>
  <Rule>
    <Title> &gt; 2477574.0 AND &lt;= 4866692.0</Title>
    <Filter>
      <And>
        <PropertyIsGreaterThan>
          <PropertyName>PERSONS</PropertyName>
          <Literal>2477574.0</Literal>
        </PropertyIsGreaterThan>
        <PropertyIsLessThanOrEqualTo>
          <PropertyName>PERSONS</PropertyName>
          <Literal>4866692.0</Literal>
        </PropertyIsLessThanOrEqualTo>
      </And>
    </Filter>
    <PolygonSymbolizer>
      <Fill>
        <CssParameter name="fill">#AA0055</CssParameter>
      </Fill>
      <Stroke/>
    </PolygonSymbolizer>
  </Rule>
  <Rule>
    <Title> &gt; 4866692.0</Title>
    <Filter>
      <PropertyIsGreaterThan>
        <PropertyName>PERSONS</PropertyName>
        <Literal>4866692.0</Literal>
      </PropertyIsGreaterThan>
    </Filter>
    <PolygonSymbolizer>
```

```
      <Fill>
        <CssParameter name="fill">#5500AA</CssParameter>
      </Fill>
      <Stroke/>
    </PolygonSymbolizer>
  </Rule>
</Rules>
```

15.21.3 Classify Raster Data

This resource is deprecated, as the classify endpoint can now handle also raster data

`/rasterize[.<format>]`

Method	Action	Status code	Formats	Default Format
GET	Create a ColorMap SLD for the given layer (of coverage type)	200	HTML, XML, JSON, SLD	HTML

The service can be used to create a ColorMap SLD for the given coverage, specyfing the **type of ColorMap** (VALUES, INTERVALS, RAMP) and one of the **predefined color ranges** (RED, BLUE, GRAY, JET, RANDOM, CUSTOM).

Using the **CUSTOM** ColorMap, startColor and endColor (and optionally midColor) have to be specified.

The parameters usable to customize the ColorMap are:

Parameter	Description	Values	Default Value
min	Minimum value for classification	double numeric value	0.0
max	Maximum value for classification	double numeric value	100.0
classes	Number of classes for the created map	integer numeric value	100
digits	Number of fractional digits for class limits (in labels)	integer numeric value	5
type	ColorMap type	INTERVALS, VALUES, RAMP	RAMP
ramp	ColorMap color ranges	RED, BLUE, GRAY, JET, RANDOM, CUSTOM	RED
startColor	starting color for the CUSTOM ramp		
endColor	ending color for the CUSTOM ramp		
midColor	central color for the CUSTOM ramp		
cache	append caching headers to the responses	expire time in seconds, use 0 to disable cache	600 (10 minutes)

Examples

A RED color ramp with 5 classes

```
curl -v -u admin:geoserver -XGET
  http://localhost:8080/geoserver/rest/sldservice/sfdem/rasterize.sld?min=0&max=100&
↪classes=5&type=RAMP&ramp=RED&digits=1
```

```
<?xml version="1.0" encoding="UTF-8"?>
<sld:StyledLayerDescriptor xmlns="http://www.opengis.net/sld" xmlns:sld="http://www.
↪opengis.net/sld" xmlns:ogc="http://www.opengis.net/ogc" xmlns:gml="http://www.
↪opengis.net/gml" version="1.0.0">
    <sld:NamedLayer>
        <sld:Name>Default Styler</sld:Name>
        <sld:UserStyle>
            <sld:Name>Default Styler</sld:Name>
            <sld:FeatureTypeStyle>
                <sld:Name>name</sld:Name>
                <sld:FeatureTypeName>gray</sld:FeatureTypeName>
                <sld:Rule>
                    <sld:RasterSymbolizer>
                        <sld:ColorMap>
                            <sld:ColorMapEntry color="#000000" opacity="0" quantity="-
↪1.0E-9" label="transparent"/>
                            <sld:ColorMapEntry color="#420000" opacity="1.0" quantity=
↪"0.0" label="0.0"/>
                            <sld:ColorMapEntry color="#670000" opacity="1.0" quantity=
↪"25.0" label="25.0"/>
                            <sld:ColorMapEntry color="#8B0000" opacity="1.0" quantity=
↪"50.0" label="50.0"/>
                            <sld:ColorMapEntry color="#B00000" opacity="1.0" quantity=
↪"75.0" label="75.0"/>
                            <sld:ColorMapEntry color="#D40000" opacity="1.0" quantity=
↪"100.0" label="100.0"/>
                        </sld:ColorMap>
                    </sld:RasterSymbolizer>
                </sld:Rule>
            </sld:FeatureTypeStyle>
        </sld:UserStyle>
    </sld:NamedLayer>
</sld:StyledLayerDescriptor>
```

A CUSTOM color ramp with 5 classes, with colors ranging from RED (0xFF0000) to BLUE (0x0000FF).

```
curl -v -u admin:geoserver -XGET
  http://localhost:8080/geoserver/rest/sldservice/sfdem/rasterize.sld?min=0&max=100&
↪classes=5&type=RAMP&ramp=CUSTOM&digits=1&startColor=0xFF0000&endColor=0x0000FF
```

```
<?xml version="1.0" encoding="UTF-8"?>
<sld:StyledLayerDescriptor xmlns="http://www.opengis.net/sld" xmlns:sld="http://www.
↪opengis.net/sld" xmlns:ogc="http://www.opengis.net/ogc" xmlns:gml="http://www.
↪opengis.net/gml" version="1.0.0">
    <sld:NamedLayer>
        <sld:Name>Default Styler</sld:Name>
        <sld:UserStyle>
            <sld:Name>Default Styler</sld:Name>
            <sld:FeatureTypeStyle>
                <sld:Name>name</sld:Name>
                <sld:FeatureTypeName>gray</sld:FeatureTypeName>
                <sld:Rule>
                    <sld:RasterSymbolizer>
                        <sld:ColorMap>
                            <sld:ColorMapEntry color="#000000" opacity="0" quantity="-
↪1.0E-9" label="transparent"/>
```

```
                        <sld:ColorMapEntry color="#FF0000" opacity="1.0" quantity=
↪"0.0" label="0.0"/>
                        <sld:ColorMapEntry color="#CC0033" opacity="1.0" quantity=
↪"25.0" label="25.0"/>
                        <sld:ColorMapEntry color="#990066" opacity="1.0" quantity=
↪"50.0" label="50.0"/>
                        <sld:ColorMapEntry color="#660099" opacity="1.0" quantity=
↪"75.0" label="75.0"/>
                        <sld:ColorMapEntry color="#3300CC" opacity="1.0" quantity=
↪"100.0" label="100.0"/>
                    </sld:ColorMap>
                </sld:RasterSymbolizer>
            </sld:Rule>
        </sld:FeatureTypeStyle>
    </sld:UserStyle>
  </sld:NamedLayer>
</sld:StyledLayerDescriptor>
```

15.22 Geofence Plugin

GeoFence offers an alternative to the GeoServer *Security* subsystem of GeoServer, allowing far more advanced security configurations, such as rules that combine data and service restrictions. It uses a client-server model, and this plugin only provides the client component. It must connect either to an external Geofence server, or be used in combination with the GeoServer integrated Geofence server *Geofence Internal Server*.

15.22.1 GeoFence promoted to extension since version 2.15

Geofence modules were community modules up to Geoserver version 2.14. Geofence modules were promoted to extensions since version 2.15 (see GSIP 164 - Promote geofence modules from Community to Extension.

Up to Geoserver version 2.14, Geofence can be downloaded from Geoserver build server.

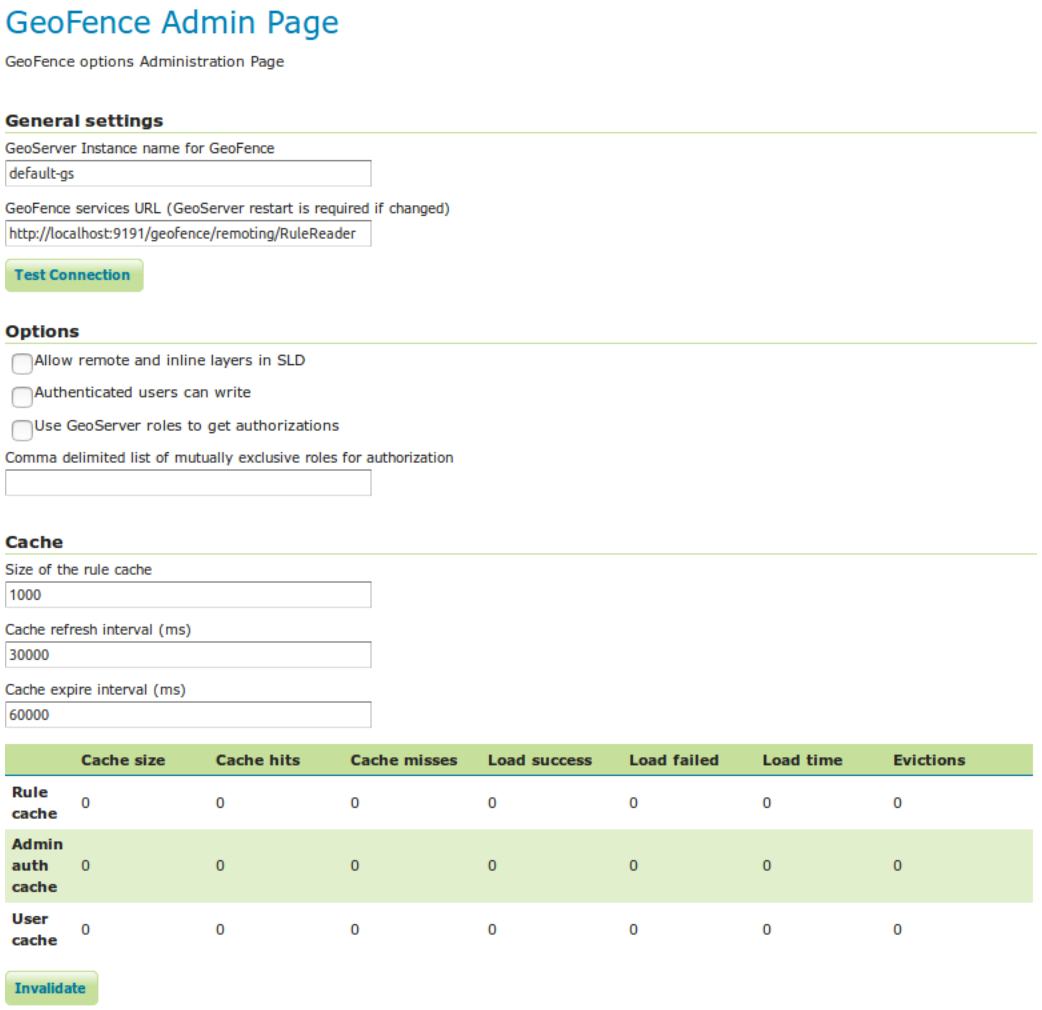
15.22.2 Installing the GeoServer GeoFence extension

For version 2.15 and later, use the standard procedure to install an extension.

1. Navigate to the GeoServer download page.
2. Find the page that matches the version of the running GeoServer.
3. Download the GeoFence extension. The download link will be in the *Extensions* section under *Other*.
4. Extract the files in this archive to the `WEB-INF/lib` directory of your GeoServer installation.
5. Restart GeoServer

15.22.3 GeoFence Admin GUI

The GeoFence Admin Page is a component of the GeoServer web interface. You can access it from the GeoServer web interface by clicking the *GeoFence* link, found on the left side of the screen after logging in.


GeoFence Admin Page
GeoFence options Administration Page
General settings
GeoServer Instance name for GeoFence
default-gs
GeoFence services URL (GeoServer restart is required if changed)
http://localhost:9191/geofence/remoting/RuleReader
Test Connection
Options
Allow remote and inline layers in SLD
Authenticated users can write
Use GeoServer roles to get authorizations
Comma delimited list of mutually exclusive roles for authorization
Cache
Size of the rule cache
1000
Cache refresh interval (ms)
30000
Cache expire interval (ms)
60000
Cache size
Cache hits
Cache misses
Load success
Load failed
Load time
Evictions
Rule cache 0 0 0 0 0 0 0
Admin auth cache 0 0 0 0 0 0 0
User cache 0 0 0 0 0 0 0
Invalidate

General Settings

Configure the following settings here:

- Geoserver instance name: the name under which this geoserver is known by the geofence server. This useful for when you use an external geofence server with multiple geoserver servers.
- GeoServer services URL: this is how geoserver knows how to connect to the external geofence server. When using an internal geofence server, this is not configurable. For example "http://localhost:9191/geofence/remoting/RuleReader" for an external geofence server on localhost.

Options

Configure the following settings here:

- Allow remote and inline layers in SLD
- Authenticated users can write
- Use GeoServer roles to get authorizations
- Comma delimited list of mutually exclusive roles for authorization

Cache

Configure the following settings here:

- Size of the rule cache (amount of entries)
- Cache refresh interval (ms)
- Cache expire interval (ms)

Collected data about the cache can be retrieved here. Per cache (rules, admin rules and users) we retrieve the cache size, hits, misses, load successes, load failures, load times and evictions. The cache can be manually invalidated (cleared).

Basic GeoServer configuration

- Login with the default administrative credentials `admin / geoserver` (or whatever you have configured before).
- In the security panel you'll find the GeoFence link to the GeoFence security admin page
- Open the GeoFence admin page; you'll get to this page:

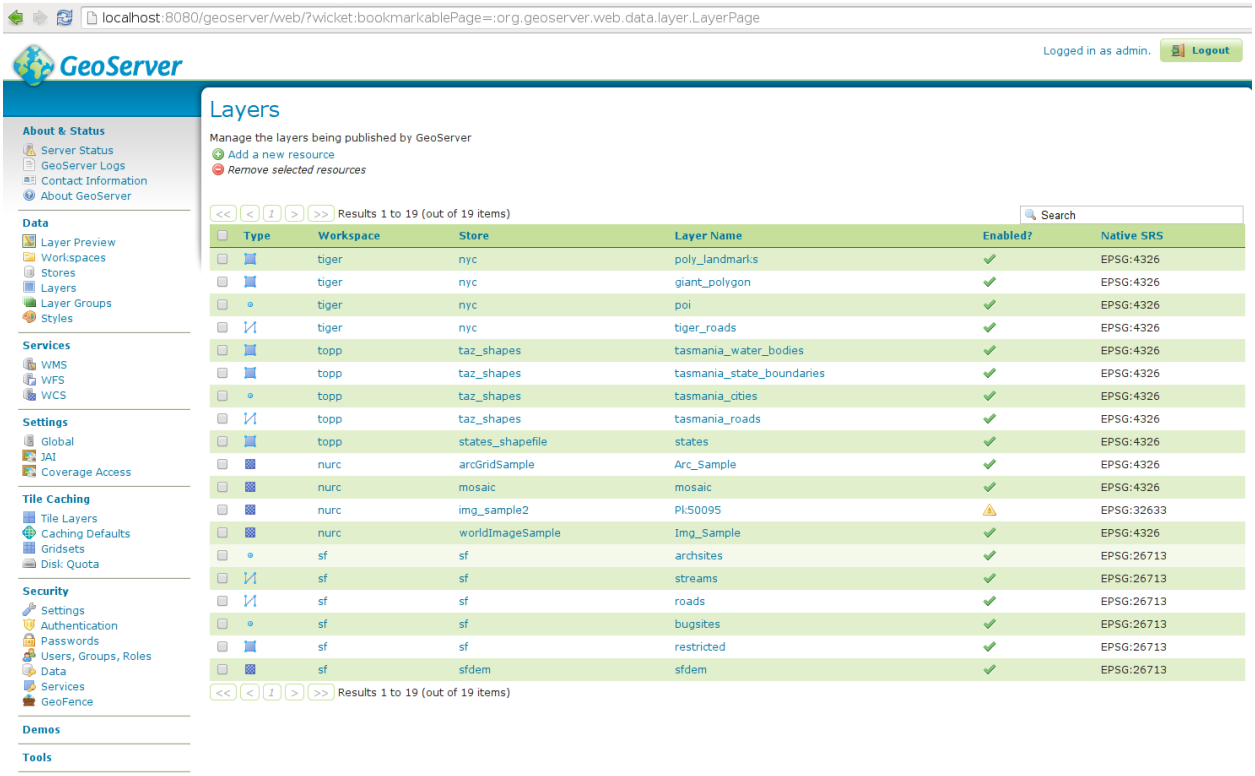
  You can notice here the information that allow the GeoFence probe inside GeoServer to communicate with the GeoFence engine:

  - the URL that the probe shall use to communicate with GeoFence;
  - the name (default is *default-gs*) this instance will use to identify itself to GeoFence. This instance name should be equal to the one we set into GeoFence.

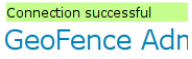
- Testing connection to GeoFence.

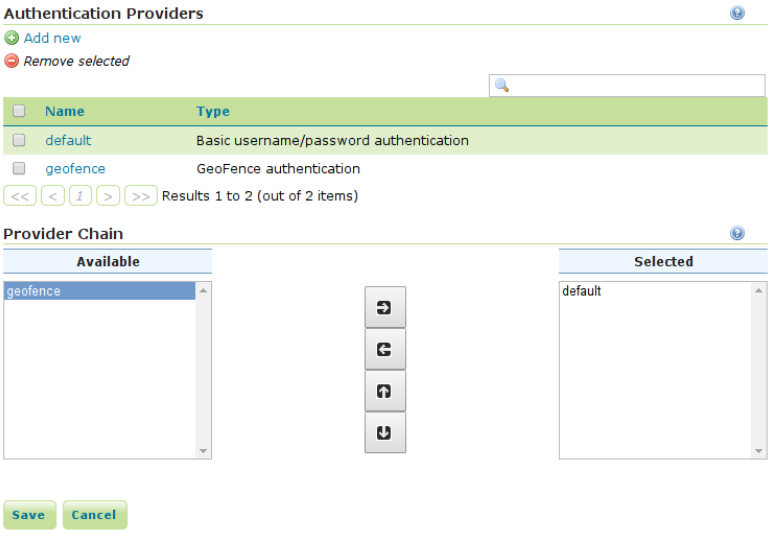
  We already performed a connection test from GeoFence to GeoServer. Using the button **Test connection** we can also test that GeoServer can communicate to GeoFence. If everything is ok, you'll get this message:

- Open the **Authentication** page under the **Security** settings:

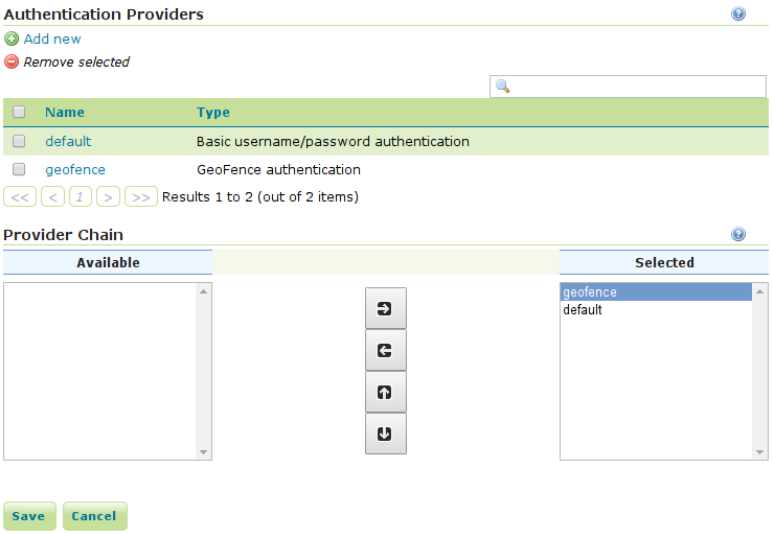

localhost:8080/geoserver/web/?wicket:bookmarkablePage=:org.geoserver.web.data.layer.LayerPage
GeoServer
Logged in as admin.
Logout
About & Status
Server Status
GeoServer Logs
Contact Information
About GeoServer
Data
Layer Preview
Workspaces
Stores
Layers
Layer Groups
Styles
Services
WMS
WFS
WCS
Settings
Global
JAI
Coverage Access
Tile Caching
Tile Layers
Caching Defaults
Gridsets
Disk Quota
Security
Settings
Authentication
Passwords
Users, Groups, Roles
Data
Services
GeoFence
Demos
Tools
Layers
Manage the layers being published by GeoServer
Add a new resource
Remove selected resources
Search
Type
Workspace
Store
Layer Name
Enabled?
Native SRS
tiger nyc poly_landmarks EPSG:4326
tiger nyc giant_polygon EPSG:4326
tiger nyc poi EPSG:4326
tiger nyc tiger_roads EPSG:4326
topp taz_shapes tasmania_water_bodies EPSG:4326
topp taz_shapes tasmania_state_boundaries EPSG:4326
topp taz_shapes tasmania_cities EPSG:4326
topp taz_shapes tasmania_roads EPSG:4326
topp states_shapefile states EPSG:4326
nurc arcGridSample Arc_Sample EPSG:4326
nurc mosaic mosaic EPSG:4326
nurc img_sample2 Pk50095 EPSG:32633
nurc worldImageSample Img_Sample EPSG:4326
sf sf archsites EPSG:26713
sf sf streams EPSG:26713
sf sf roads EPSG:26713
sf sf bugsites EPSG:26713
sf sf restricted EPSG:26713
sf sf sfdem EPSG:26713
Results 1 to 19 (out of 19 items)


Security
Settings
Authentication
Passwords
Users, Groups, Roles
Data
Services
GeoFence


Connection successful
GeoFence Adn


Authentication Providers
Add new
Remove selected
Name
Type
default
Basic username/password authentication
geofence
GeoFence authentication
Results 1 to 2 (out of 2 items)
Provider Chain
Available
Selected
geofence
default
Save
Cancel

- Add the GeoFence authenticator and **put it as the first in the list** otherwise you will not be able to login as `admin/admin`:



- Now that we added GeoFence as authentication provider, we'll be able to log into GeoServer using the credentials we added in GeoFence (user `admin` and user *tiger*). Try and log in using user `tiger`.

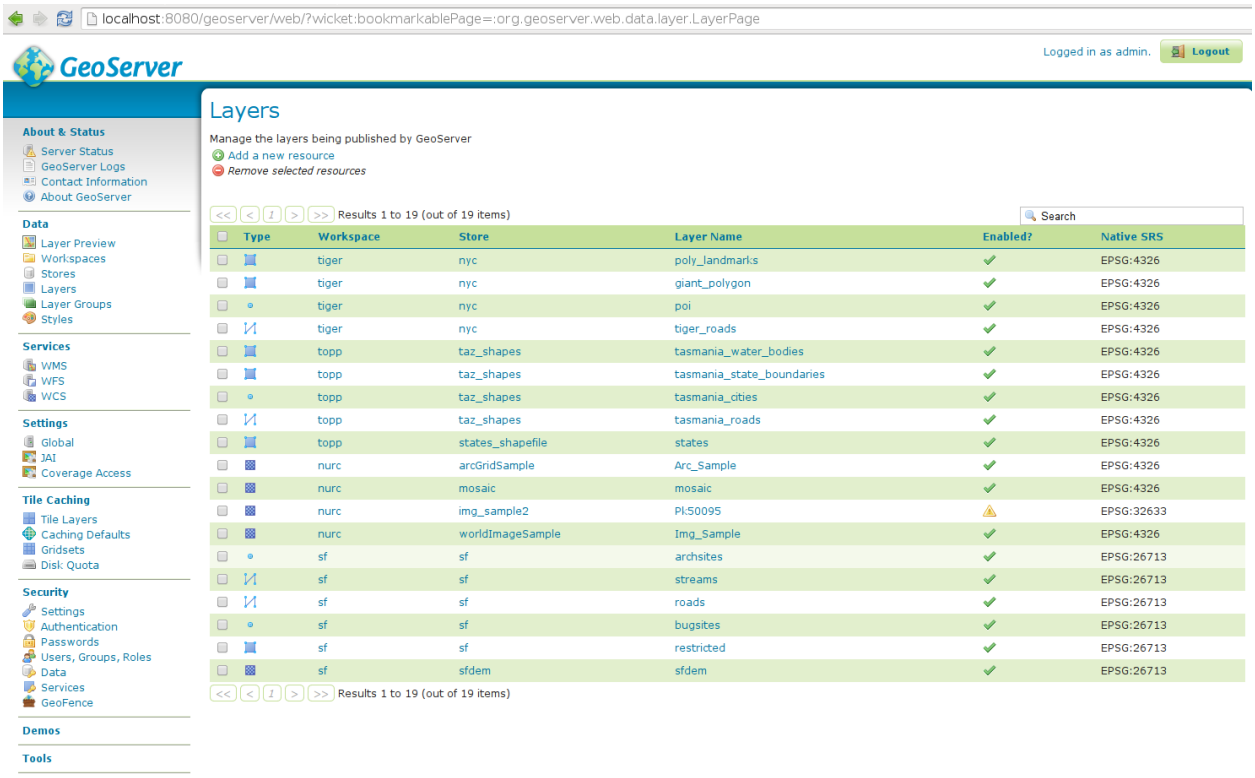
Testing authorization

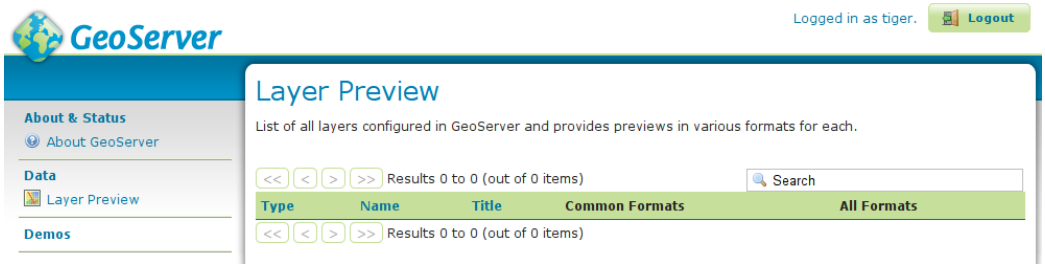
- Logging into GeoServer as admin you will be able to see all the defined layers:
- Logging into GeoServer as a non-admin user, the defined rules will be examined; since we defined no rules yet, the default behaviour is to deny access to all resources:
- Get back to GeoFence, and add a rule which allows all layers in workspace tiger for user `tiger`; create a rule defining:
  - user `tiger`
  - instance `default-gs`
  - workspace `tiger` (you will get a dropdown menu containing all the workspaces available in the selected instance)
  - grant type: `allow` You'll get a line like this one:
- Verify the new authorizations.

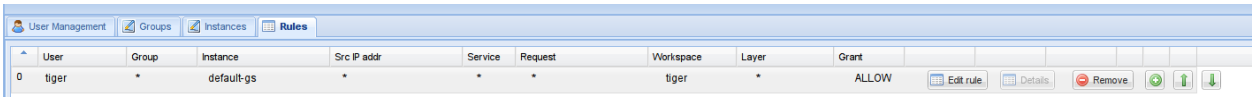
  Since the probe caches the GeoFence responses, you may need to login again as administrator (or you may keep an admin session open in another browser) and clear the probe cache. You can do it by pressing the "Invalidate" button in the bottom of the GeoFence admin page:
- Login again in GeoServer as user tiger and you will see in **layer preview** all the layers in the `tiger` workspace:

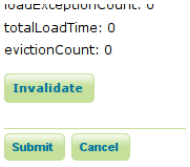
15.22.4 GeoFence Cache REST

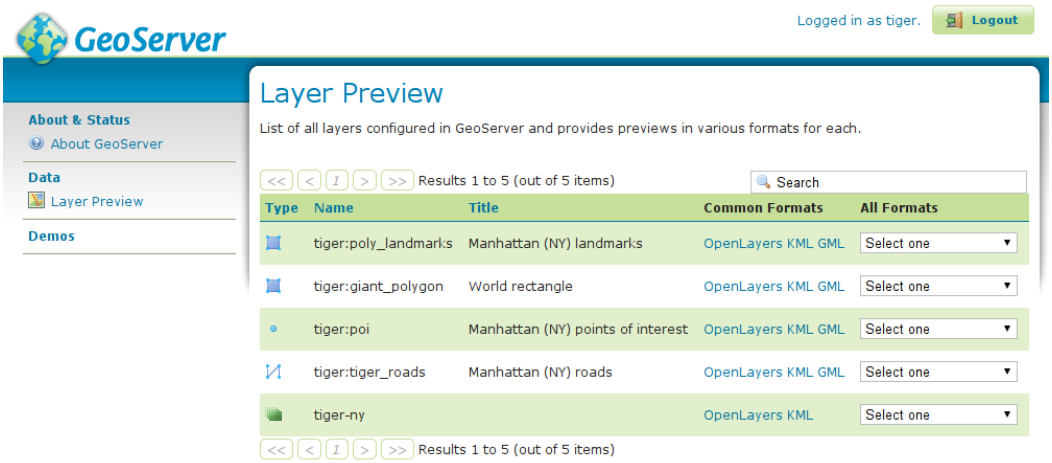
The Geofence client cache status can be queried, and the cache cleared (invalidated) through a REST service.


localhost:8080/geoserver/web/?wicket:bookmarkablePage=:org.geoserver.web.data.layer.LayerPage
GeoServer
Logged in as admin.
Logout
About & Status
Server Status
GeoServer Logs
Contact Information
About GeoServer
Data
Layer Preview
Workspaces
Stores
Layers
Layer Groups
Styles
Services
WMS
WFS
WCS
Settings
Global
JAI
Coverage Access
Tile Caching
Tile Layers
Caching Defaults
Gridsets
Disk Quota
Security
Settings
Authentication
Passwords
Users, Groups, Roles
Data
Services
GeoFence
Demos
Tools
Layers
Manage the layers being published by GeoServer
Add a new resource
Remove selected resources
Results 1 to 19 (out of 19 items)
Search
Type
Workspace
Store
Layer Name
Enabled?
Native SRS
tiger nyc poly_landmarks EPSG:4326
tiger nyc giant_polygon EPSG:4326
tiger nyc poi EPSG:4326
tiger nyc tiger_roads EPSG:4326
topp taz_shapes tasmania_water_bodies EPSG:4326
topp taz_shapes tasmania_state_boundaries EPSG:4326
topp taz_shapes tasmania_cities EPSG:4326
topp taz_shapes tasmania_roads EPSG:4326
topp states_shapefile states EPSG:4326
nurc arcGridSample Arc_Sample EPSG:4326
nurc mosaic mosaic EPSG:4326
nurc img_sample2 Pk50095 EPSG:32633
nurc worldImageSample Img_Sample EPSG:4326
sf sf archsites EPSG:26713
sf sf streams EPSG:26713
sf sf roads EPSG:26713
sf sf bugsites EPSG:26713
sf sf restricted EPSG:26713
sf sf sfdem EPSG:26713
Results 1 to 19 (out of 19 items)


GeoServer
Logged in as tiger.
Logout
About & Status
About GeoServer
Data
Layer Preview
Demos
Layer Preview
List of all layers configured in GeoServer and provides previews in various formats for each.
Results 0 to 0 (out of 0 items)
Search
Type
Name
Title
Common Formats
All Formats
Results 0 to 0 (out of 0 items)


User Management
Groups
Instances
Rules
User
Group
Instance
Src IP addr
Service
Request
Workspace
Layer
Grant
0 tiger * default-gs * * * tiger * ALLOW
Edit rule
Details
Remove


totalLoadTime: 0
evictionCount: 0
Invalidate
Submit
Cancel



Requests

`/geofence/ruleCache/info`

Retrieve information about the geofence cache status.

Method	Action	Parameters	Response
GET	Retrieve information about the geofence cache status. Per cache (rules, admin rules and users) we retrieve the cache size, hits, misses, load successes, load failures, load times and evictions.	—	200 OK. Text Format.

`/geofence/ruleCache/invalidate`

Invalidate the geofence cache.

Method	Action	Parameters	Response
PUT	Invalidate (clear) the geofence cache	—	200 OK.

15.23 Geofence Internal Server

This plugin runs a GeoFence server integrated internally in GeoServer. Geofence allows far more advanced security configurations than the default GeoServer *Security* subsystem, such as rules that combine data and service restrictions.

In the integrated version, the users and roles service configured in geoserver are associated with the geofence rule database. The integrated geofence server can be configured using its WebGUI page or REST configuration.

15.23.1 GeoFence promoted to extension since version 2.15

Geofence modules were community modules up to Geoserver version 2.14. Geofence modules were promoted to extensions since version 2.15 (see GSIP 164 - Promote geofence modules from Community to Extension.

Up to Geoserver version 2.14, Geofence can be downloaded from Geoserver build server.

15.23.2 Installing the GeoServer GeoFence Server extension

1. Navigate to the GeoServer download page.
2. Find the page that matches the version of the running GeoServer.
3. Download the GeoFence extension. The download link will be in the *Extensions* section under *Other*.
4. Extract the files in this archive to the `WEB-INF/lib` directory of your GeoServer installation.

> **Warning:** By default GeoFence will store his data in a H2 database and the database schema will be automatically managed by Hibernate. GeoFence documentation explains how to configure a different backed database and configure Hibernate behavior.

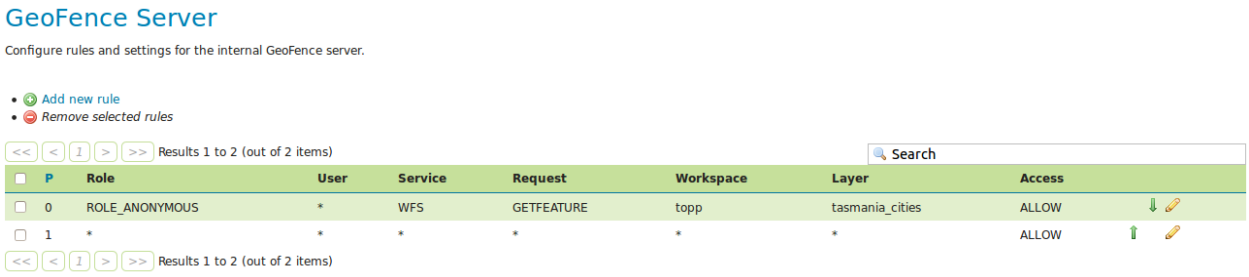
5. Restart GeoServer

15.23.3 GeoFence Server GUI

The GeoFence user interface is a component of the GeoServer web interface. You can access it from the GeoServer web interface by clicking the *GeoFence Server* link, found on the left side of the screen after logging in.

Rules page

An overview of all rules is provided with priority, the rule's scope specifications (role, user, service, request, workspace and layer) and its access behaviour. The '*' symbol means that the rule applies to all possible values of that specification. Rules are always ordered by priority, but the order can be reversed by pressing the 'P' priority column header.

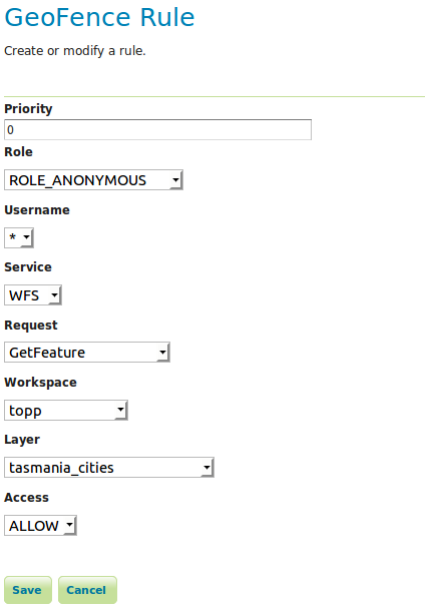


A new rule can be added with the "Add new rule" link. Any number of rules can be deleted by selecting them and then clicking on the "Remove selected rules" link.

Rule priority order can be easily on this page through the up and down arrows on the right side. Rules can be modified using the pencil symbol, which opens the rule page.

Rule page

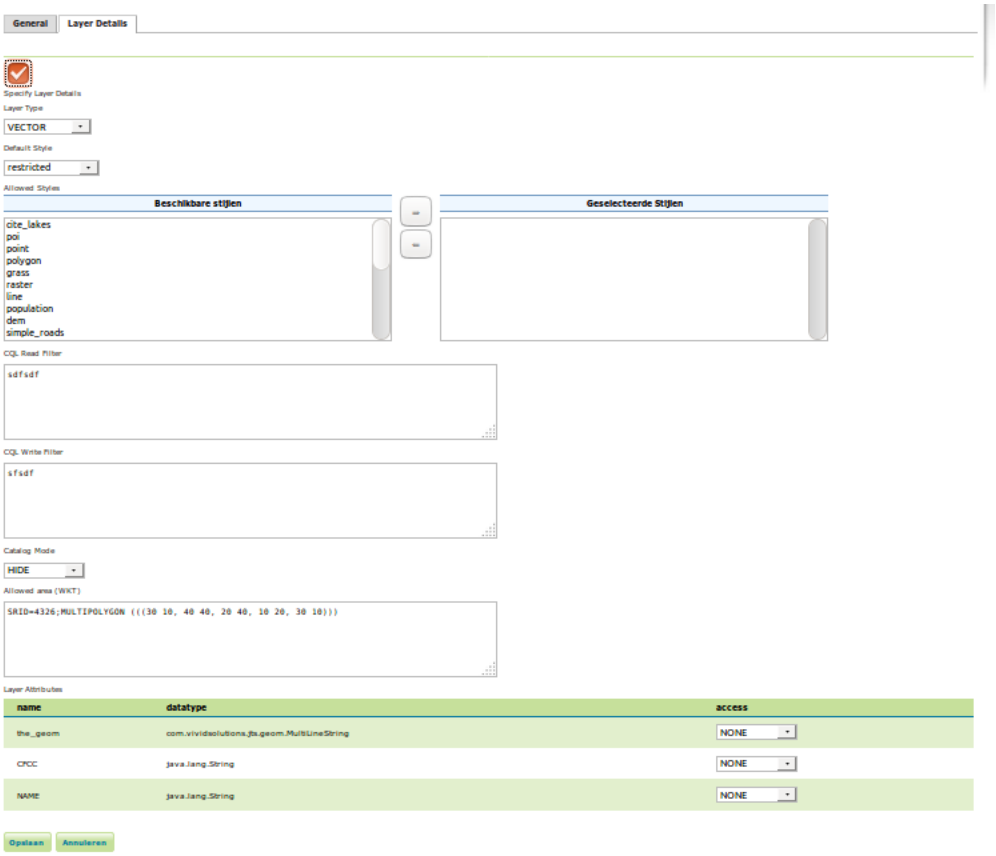
This page is displayed both when creating a new rule and modifying an existing rule.



Priority can be changed manually by specifying a priority number. If this priority number is already occupied by another rule, this will cause that rule and all rules after it to shift one place to a lower priority.

When Access type LIMIT is selected, additional options are displayed that allows the user to select the Catalog Mode and the Allowed Area (WKT) associated with this rule.

When Access type ACCESS is selected as well as a specific layer, it becomes possible to specify the "layer details" in a separate tab. These make it possible to add additional filters to the layer for the rule in question. For example, the rule can alter the default style of the layer, specify which styles are available at all, which attributes are accessible for reading and/or writing, specify CQL filters for both reading and writing, specify a catalog mode, and an allowed area (WKT) filter.


General
Layer Details
Specify Layer Details
Layer Type
VECTOR
Default Style
restricted
Allowed Styles
Beschikbare stijlen
Geselecteerde Stijlen
cite_lakes
poi
point
polygon
grass
raster
line
population
dem
simple_roads
CQL Read Filter
sdfsdf
CQL Write Filter
sfsdf
Catalog Mode
HIDE
Allowed area (WKT)
SRID=4326;MULTIPOLYGON (((30 10, 40 40, 20 40, 10 20, 30 10)))
Layer Attributes
name
datatype
access
the_geom
com.vividsolutions.jts.geom.MultiLineString
NONE
CFCC
java.lang.String
NONE
NAME
java.lang.String
NONE
Opslaan
Annuleren

15.23.4 GeoFence Rest API

Security

The Geofence Rest API is only accessible to users with the role ROLE_ADMIN.

Input/Output

Data Object Transfer

Both XML and JSON are supported for transfer of data objects. The default is XML. Alternatively, JSON may be used by setting the 'content-type' (POST) and 'accept' (GET) http headers to 'application/json' in your requests.

Encoding of a rule in XML:

```
<Rule>
        <id>..</id>
        <priority>..</priority>
        <userName>..</userName>
        <roleName>..</roleName>
        <workspace>..</workspace>
        <layer>..</layer>
        <service>..</service>
        <request>..</request>
        <access> ALLOW | DENY | LIMIT </access>

        <!-- for LIMIT access rules-->
        <limits>
                <allowedArea>..</allowedArea>
                <catalogMode> HIDE | CHALLENGE | MIXED </catalogMode>
        </limits>

        <!-- for ALLOW access rules with specified layer -->
        <layerDetails>
                <layerType> VECTOR | RASTER | LAYERGROUP </layerType>

                <defaultStyle>..</defaultStyle>

                <cqlFilterRead>..</cqlFilterRead>

                <cqlFilterWrite>..</cqlFilterWrite>

                <allowedArea>..</allowedArea>

                <catalogMode> HIDE | CHALLENGE | MIXED </catalogMode>

                <allowedStyle>..</allowedStyle>

                <allowedStyle>..</allowedStyle>

                ..

                <attribute>
                        <name>..</name>
                        <datatype>..</datatype>
                        <accessType> NONE | READONLY | READWRITE </accessType>
```

```
                </attribute>

                <attribute>
                        <name>..</name>
                        <datatype>..</datatype>
                        <accessType> NONE | READONLY | READWRITE </accessType>
                </attribute>

                ..

        </layerDetails>
</Rule>
```

Encoding of a rule in JSON:

```
{"Rule": {"id":..,"priority":..,"userName":"..","roleName":"..","workspace":"..",
↪"layer":"..","service":"..","request":"..","access":".."}}
```

In case a rule that has "any" ("*") for a particular field the field is either not included (default), left empty or specified with a single asterisk (the latter two may be used for updates to distinguish from "do not change this field").

Encoding of a list of rules in XML:

```
<Rules count="n">
        <Rule> ... </Rule>
        <Rule> ... </Rule>
        ...
</Rules>
```

The result of a count would not include the actual <Rule> tags.

Encoding of a list of rules in JSON:

```
{"count":n,"rules":[{..},{..},..]}
```

Filter Parameters

All filter parameters are optional.

Name	Type	Description
page	number	Used for paging a list of rules. Specifies the number of the page. Leave out for no paging. If specified, `entries` should also be specified.
entries	number	Used for paging a list of rules. Specifies the number of entries per page. Leave out for no paging. If specified, `page` should also be specified.
userName	string	Filter rules on username (excludes all other specified usernames).
userAny	0 or 1.	Specify whether rules for 'any' username are included or not.
roleName	string	Filter rules on rolename (excludes all other specified rolenames).
roleAny	0 or 1.	Specify whether rules for 'any' rolename are included or not.
service	string	Filter rules on service (excludes all other specified services).
serviceAny	0 or 1.	Specify whether rules for 'any' service are included or not.
request	string	Filter rules on request (excludes all other specified requests).
requestAny	0 or 1.	Specify whether rules for 'any' request are included or not.
workspace	string	Filter rules on workspace (excludes all other specified workspaces).
workspaceAny	0 or 1.	Specify whether rules for 'any' workspace are included or not.
layer	string	Filter rules on layer (excludes all other specified layers).
layerAny	0 or 1.	Specify whether rules for 'any' layer are included or not.

Requests

`/rest/geofence/rules/`

Query all rules or add a new rule.

Method	Action	Supported parameters	Response
GET	List all rules, with respect to any added filters	page, entries, userName, userAny, roleName, roleAny, service, serviceAny, request, requestAny, workspace, workspaceAny, layer, layerAny	200 OK. List of rules in XML.
POST	Add a new rule	None	201 Inserted. Created `ID` header.

`/rest/geofence/rules/count`

Counts (filtered) rules.

Method	Action	Supported parameters	Response
GET	Count all rules, with respect to any added filters	userName, userAny, roleName, roleAny, service, serviceAny, request, requestAny, workspace, workspaceAny, layer, layerAny	200 OK. Rule list count in XML.

/rest/geofence/rules/id/<id>

Query, modify or delete a specific rule.

Method	Action	Supported parameters	Response
GET	Read rule information	None	200 OK. Rule in XML.
POST	Modify the rule, unspecified fields remain unchanged.	None	200 OK.
DELETE	Delete the rule	None	200 OK.

15.23.5 AdminRules Rest API

Security

The Geofence Rest API is only accessible to users with the role ROLE_ADMIN.

Input/Output

Data Object Transfer

Both XML and JSON are supported for transfer of data objects. The default is XML. Alternatively, JSON may be used by setting the 'content-type' (POST) and 'accept' (GET) http headers to 'application/json' in your requests.

Encoding of an AdminRule in XML:

```
<AdminRule>
        <id>..</id>
        <priority>..</priority>
        <userName>..</userName>
        <roleName>..</roleName>
        <addressRange>..</addressRange>
        <workspace>..</workspace>
        <access>..</access>
</AdminRule>
```

Encoding of a rule in JSON:

```
{"id":..,"priority":..,"userName":"..","roleName":"..","addressRange":"..","workspace
↪":"..","access":".."}
```

In case a rule that has "any" ("*") for a particular field the field is either not included (default), left empty or specified with a single asterisk (the latter two may be used for updates to distinguish from "do not change this field").

`access` should be either `ADMIN` or `USER`.

`addressRange` is a string in CIDR notation (block/bits: e.g. `127.0.0.1/32`).

Encoding of a list of rules in XML:

```
<AdminRules count="n">
        <AdminRule> ... </AdminRule>
        <AdminRule> ... </AdminRule>
        ...
</AdminRules>
```

The result of a count would not include the actual <AdminRule> tags.

Encoding of a list of rules in JSON:

```
{"count":n,"adminrules":[{..},{..},..]}
```

Filter Parameters

All filter parameters are optional.

Name	Type	Description
page	number	Used for paging a list of rules. Specifies the number of the page. Leave out for no paging. If specified, `entries` should also be specified.
entries	number	Used for paging a list of rules. Specifies the number of entries per page. Leave out for no paging. If specified, `page` should also be specified.
userName	string	Filter rules on username (excludes all other specified usernames).
userAny	0 or 1.	Specify whether rules matching any username should be included or not.
roleName	string	Filter rules on rolename (excludes all other specified rolenames).
roleAny	0 or 1.	Specify whether rules matching any rolename should be included or not.
workspace	string	Filter rules on workspace (excludes all other specified workspaces).
workspaceAny	0 or 1.	Specify whether rules matching any workspace should be included or not.

Requests

`/geofence/rest/adminrules/`

Query all adminrules or add a new adminrule.

Method	Action	Supported parameters	Response
GET	List all adminrules, with respect to any added filters	page, entries, userName, userAny, roleName, roleAny, workspace, workspaceAny	200 OK. List of adminrules in XML.
POST	Add a new adminrule	None	201 Inserted. Created `ID` header.

`/geofence/rest/adminrules/count`

Counts (filtered) adminrules.

Method	Action	Supported parameters	Response
GET	Count all adminrules, with respect to any added filters	userName, userAny, roleName, roleAny, workspace, workspaceAny	200 OK. Rule list count in XML.

`/geofence/rest/adminrules/id/<id>`

Query, modify or delete a specific adminrule.

Method	Action	Supported parameters	Response
GET	Read adminrule information	None	200 OK. AdminRule in XML.
POST	Modify the adminrule, unspecified fields remain unchanged.	None	200 OK.
DELETE	Delete the adminrule	None	200 OK.

15.23.6 Using the Internal GeoFence server (Tutorial)

Introduction

This tutorial shows how to install and configure the *Geofence Internal Server* plug-in. It shows how to create rules in two ways: using the GUI and REST methods.

The tutorial assumes:

- GeoServer is running on http://localhost:8080/geoserver
- You have a user/group service called "default" that allows the creation of new users. If your primary user/group service is not called "default", you must start geoserver with the following java system property present:

```
org.geoserver.rest.DefaultUserGroupServiceName=<name_of_usergroupservice>
```

with <name_of_usergroupservice> a user/group service that allows the creation of new users.

Getting Started

Install the plugin-in, see *GeoFence promoted to extension since version 2.15*. Configure the user/group service as described above if necessary.

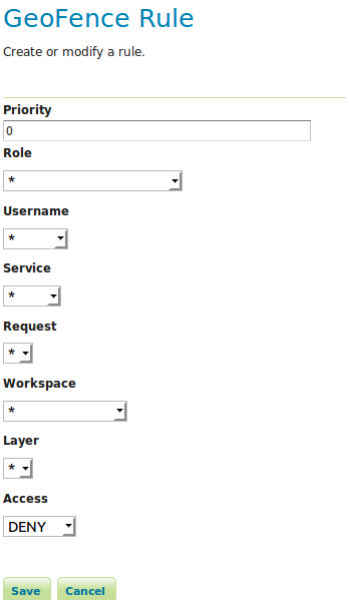
Restart GeoServer.

> **Note:** Since we defined no rules yet, the default behavior of GeoFence is to deny access to all resources.

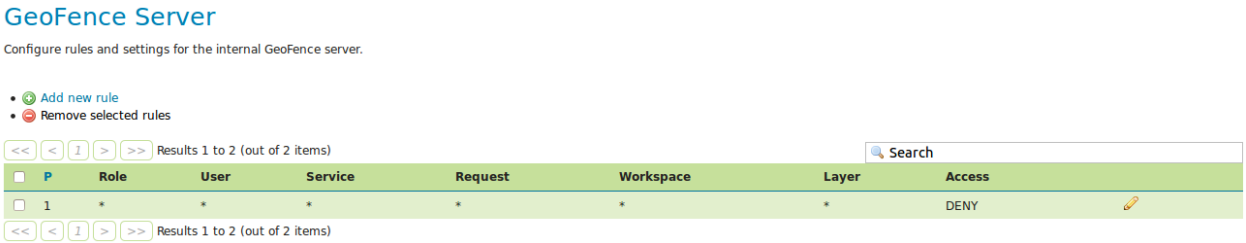
There should now be a *GeoFence Server* link on the left side of the screen after logging in. Click on it. This is the configuration page of your internal GeoFence.

Creating new Rules with the GUI

1. Click on the "Add new rule" link. Change only "Access" to "DENY".



Click on "Save".



We have now expressed that the first rule (with lowest priority) disallows everyone from everything. The following more specific rules we make will provide the exceptions to that general rule. It is also possible to do it the other way (allow everyone to anything as most general rule and specify exceptions to that.)

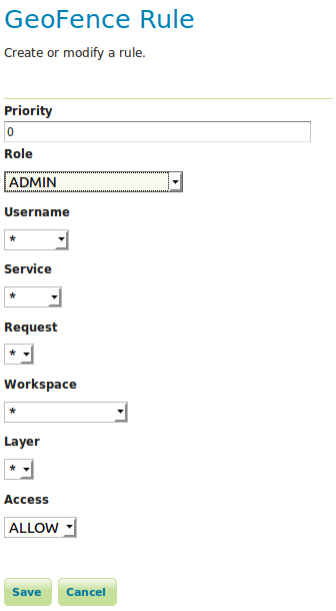
2. As a next step, we will grant the administrator access to everything. Click on "Add new rule" again. Change "Role" to "ADMIN" and click "Save".

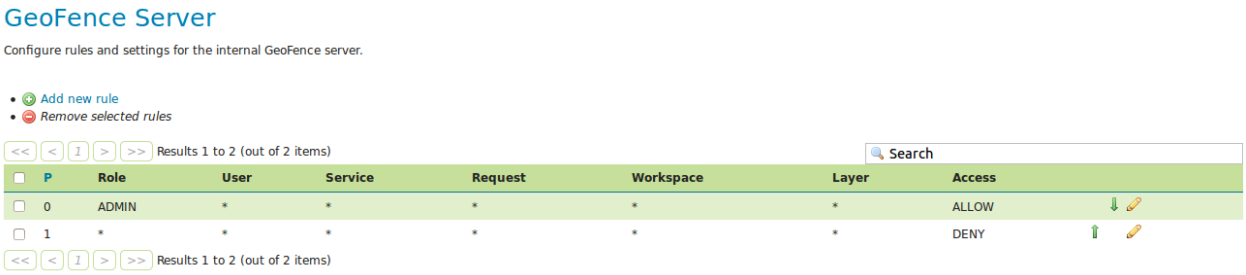
You now have a working, basic security configuration.

Creating rules with the REST API

1. Open a new tab with your browser and go to the following URL: http://localhost:8080/geoserver/geofence/rest/rules. You should get an XML representation of your rules:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Rules count="2">
      <Rule id="2">
              <access>ALLOW</access>
              <priority>0</priority>
```


GeoFence Rule
Create or modify a rule.
Priority
0
Role
ADMIN
Username
*
Service
*
Request
*
Workspace
*
Layer
*
Access
ALLOW
Save
Cancel


GeoFence Server
Configure rules and settings for the internal GeoFence server.
Add new rule
Remove selected rules
Results 1 to 2 (out of 2 items)
Search
P
Role
User
Service
Request
Workspace
Layer
Access
0
ADMIN
*
*
*
*
*
ALLOW
1
*
*
*
*
*
*
DENY
Results 1 to 2 (out of 2 items)

```
                <roleName>ADMIN</roleName>
        </Rule>
        <Rule id="1">
                <access>DENY</access>
                <priority>1</priority>
        </Rule>
</Rules>
```

2. Let us first create a new user. Do this by sending a POST request to the following URL http://localhost:8080/geoserver/rest/security/usergroup/users with the following content:

```
<user>
        <userName>michaeljfox</userName>
        <password>back2$future</password>
        <enabled>true</enabled>
</user>
```

You should receive a `201 Created` HTTP Response.

3. Now we will create an access rule for this user. Do this by sending a POST request to the following URL: http://localhost:8080/geoserver/geofence/rest/rules with the following content:

```
<Rule>
        <userName>michaeljfox</userName>
        <workspace>topp</workspace>
        <layer>states</layer>
        <service>WMS</service>
        <request>GetMap</request>
        <access>ALLOW</access>
</Rule>
```

Again, you should receive a `201 Created` HTTP Response. When browsing to the URL http://localhost:8080/geoserver/geofence/rest/rules we should now see the following information:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Rules count="2">
        <Rule id="3">
                <access>ALLOW</access>
                <layer>states</layer
                <priority>0</priority>
                <request>GETMAP</request>
                <service>WMS</service>
                <userName>michaeljfox</userName>
                <workspace>topp</workspace>
        </Rule>
        <Rule id="2">
                <access>ALLOW</access>
                <priority>0</priority>
                <roleName>ADMIN</roleName>
        </Rule>
        <Rule id="1">
                <access>DENY</access>
                <priority>1</priority>
        </Rule>
</Rules>
```

4. It should now be possible to log on with username `michaeljfox` and password `back2$future` and perform a `GetMap` on the layer `topp:states`, but nothing else.

15.23.7 Migrating old GeoFence configuration to GeoServer 2.12 and following

Starting from GeoServer 2.12, the `allowDynamicStyles` GeoFence configuration option has been moved to the core GeoServer WMS module.

This means that if you had this option active in GeoFence, you have to manually enable the same option in the WMS service configuration page of the GeoServer Admin UI (either globally or on a virtual service by virtual service basis).

See here: *WMS settings*

CHAPTER 16

Community modules

This section is devoted to GeoServer community modules. Community modules are considered "pending" in that they are not officially part of the GeoServer releases. They are however built along with the nightly builds, so you can download and play with them.

Warning: Community modules are generally considered experimental in nature and are often under constant development. For that reason documentation in this section should not be considered solid or final and will be subject to change.

16.1 Key authentication module

The `authkey` module for GeoServer allows for a very simple authentication protocol designed for OGC clients that cannot handle any kind of security protocol, not even the HTTP basic authentication.

For these clients the module allows a minimal form of authentication by appending a unique key in the URL that is used as the sole authentication token. Obviously this approach is open to security token sniffing and must always be associated with the use of HTTPS connections.

A sample authenticated request looks like:

```
http://localhost:8080/geoserver/topp/wms?service=WMS&version=1.3.0&
↪request=GetCapabilities&authkey=ef18d7e7-963b-470f-9230-c7f9de166888
```

Where `authkey=ef18d7e7-963b-470f-9230-c7f9de166888` is associated to a specific user (more on this later). The capabilities document contains backlinks to the server itself, linking to the URLs that can be used to perform GetMap, GetFeatureInfo and so on. When the `authkey` parameter is provided the backlinks will contain the authentication key as well, allowing any compliant WMS client to access secured resources.

16.1.1 Limitations

The `authkey` module is meant to be used with OGC services. It won't work properly against the administration GUI, nor RESTConfig.

16.1.2 Key providers

Key providers are responsible for mapping the authentication keys to a user. The authentication key itself is a UUID (Universal unique Identifier). A key provider needs a user/group service and it is responsible for synchronizing the authentication keys with the users contained in this service.

Key provider using user properties

This key provider uses a user property `UUID` to map the authentication key to the user. User properties are stored in the user/group service. Synchronizing is simple since the logic has to search for users not having the property `UUID` and add it. The property value is a generated UUID.

```
UUID=b52d2068-0a9b-45d7-aacc-144d16322018
```

If the user/group service is read only, the property has to be added from outside, no synchronizing is possible.

Key provider using a property file

This key provider uses a property file named `authkeys.properties`. The default user/group service is named `default`. The `authkeys.properties` for this service would be located at

`$GEOSERVER_DATA_DIR/security/usergroup/default/authkeys.propeties`

A sample file looks as follows:

```
# Format is authkey=username
b52d2068-0a9b-45d7-aacc-144d16322018=admin
1825efd3-20e1-4c18-9648-62c97d3ee5cb=sf
ef18d7e7-963b-470f-9230-c7f9de166888=topp
```

This key provider also works for read only user/group services. Synchronizing adds new users not having an entry in this file and removes entries for users deleted in the user/group service.

Key provider using an external web service

This key provider calls an external URL to map the authentication key to the user. This allows GeoServer to integrate into an existing security infrastructure, where a session token is shared among applications and managed through dedicated web services.

The web service URL and some other parameters can be specified to configure the mapper in details:

Option	Description
`Web Service URL, with key placeholder`	the complete URL of the key mapping webservice, with a special placeholder ({key}) that will be replaced by the current authentication key
`Web Service Response User Search Regular Expression`	a regular expression used to extract the username from the webservice response; the first matched group (the part included in parentheses) in the regular expression will be used as the user name; the default (^\s*(.*)\s*$) takes all the response text, trimming spaces at both ends
`Connection Timeout`	timeout to connect to the webservice
`Read Timeout`	timeout to read data from the webservice

The mapper will call the webservice using an HTTP GET request (webservice requiring POST are not supported yet), replacing the {key} placeholder in the configured URL with the actual authentication key.

If a response is received, it is parsed using the configured regular expression to extract the user name from it. New lines are automatically stripped from the response. Some examples of regular expression that can be used are:

Regular Expression	Usage
`^\\s*(.*)\\s*$`	all text trimming spaces at both ends
`^.*?\"user\"\\s*:\\s*\"([^\"]+)\".*$`	json response where the username is contained in a property named **user**
`^.*?<username>(.*?)</username>.*$`	xml response where the username is contained in a tag named **username**

Synchronizing users with the user/group service is not supported by this mapper.

AuthKEY WebService Body Response UserGroup Service

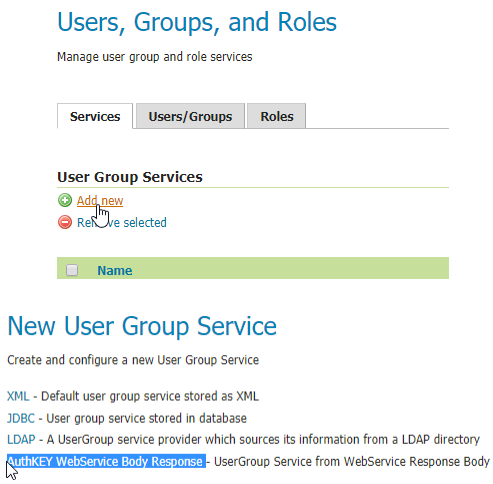
When using an external web service to get Auth details, it is possible to define a custom `GeoServer UserGroup Service` being able to fetch Authorities - aka user's Roles - from the HTTP Body Response.

The rationale is mostly the same; that kind of `GeoServer UserGroup Service` will be apply a `rolesRegex` - Roles Regular Expression - to the body response - which can be either XML, JSON or Plain Text/HTML - in order to fetch the list of available Authorities.

In order to do this, it is possible to configure instances of **AuthKEY WebService Body Response** User Group Service.

First thing to do is to:

1. Login as an `Administrator`
2. Move to `Security` > `Users, Groups, Roles` and select `Add new` from `User Group Services`
3. Click on `AuthKEY WebService Body Response`
4. Provide a `Name` and select anything you want from `Passwords` - those won't be used by this service, but they are still mandatory for GeoServer -



5. Provide a suitable `Roles Regex` to apply to your Web Service Response

> **Note:** This is the only real mandatory value to provide. The others are optional and will allow you to customize the User Group Service behavior (see below)

Once the new `GeoServer UserGroup Service` has been configured, it can be easily linked to the `Key Provider Web Service Mapper`.

1. From `Authentication` > `Authentication Filters`, select - or add new - `AuthKEY` using `Web Service` as key mapper
2. Select the newly defined `UserGroup Service` and save

Additional Options

1. *Optional static comma-separated list of available Groups from the Web Service response*

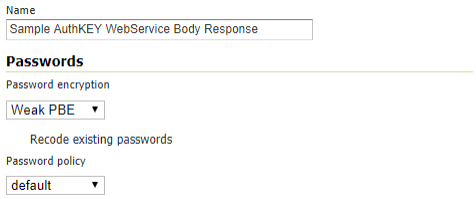
   It is worth notice that this `UserGroup Service` will **always** translate fetched Roles in the form `ROLE_<ROLENAME>`

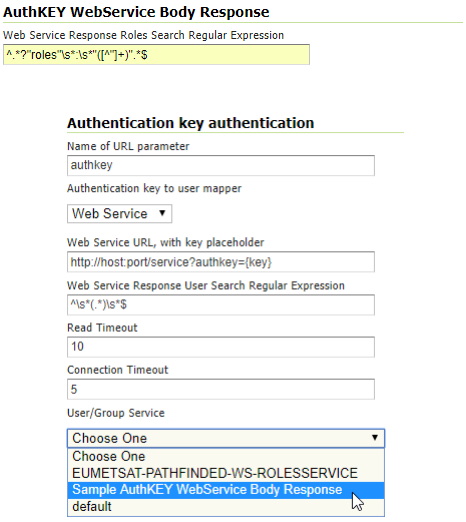
   As an instance, if the `Roles Regular Expression` will match something like:

```
my_user_role1, another_custom_user_role, role_External_Role_X
```

   this will be converted into **3** different `GeoServer User Roles` named as:

```
ROLE_MY_USER_ROLE1
ROLE_ANOTHER_CUSTOM_USER_ROLE
ROLE_EXTERNAL_ROLE_X
```

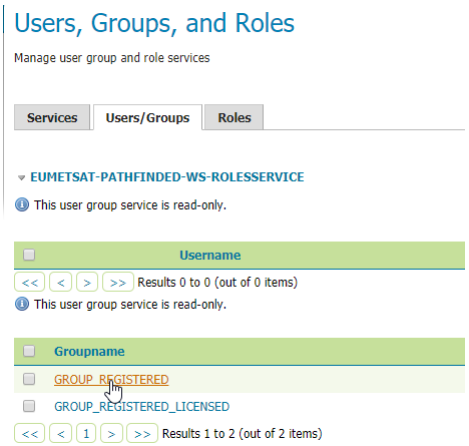




Of course the role names are known only at runtime; nevertheless it is possible to **statically** specify associated `GeoServer User Groups` to be mapped later to other internal `GeoServer User Roles`.

What does this means? A `GeoServer User Group` can be defined on the GeoServer Catalog and can be mapped by the active `Role Services` to one or more specific `GeoServer User Roles`.

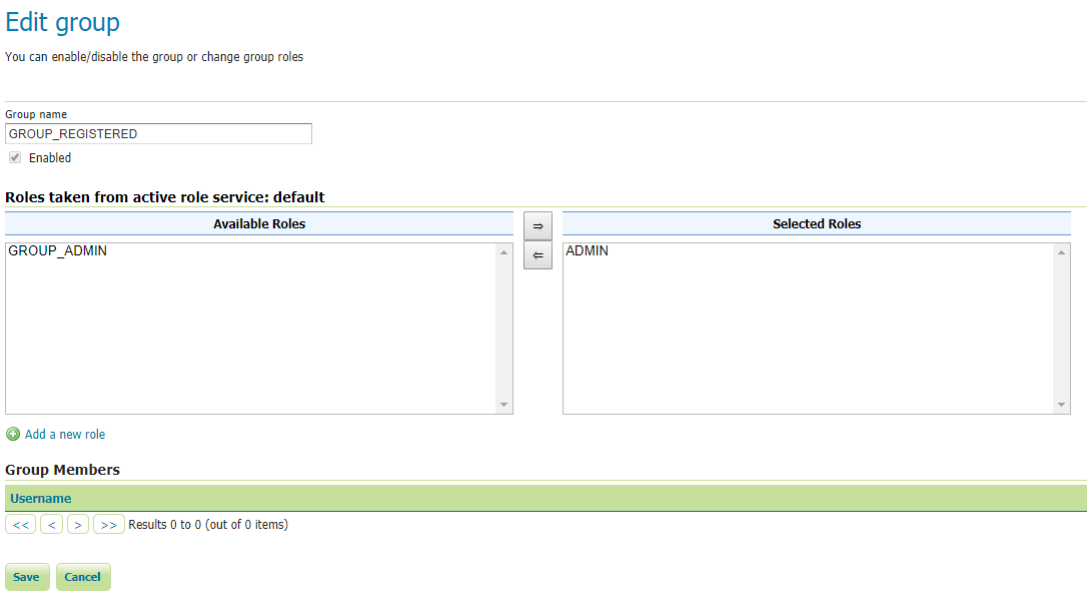
This mainly depends on the `GeoServer Role Service` you use. By default, the internal `GeoServer Role Service` can map Roles and Groups through static configuration stored on the GeoServer Data Dir. This is possible by editing `GeoServer User Group` details from the `Users, Groups, and Roles` panel



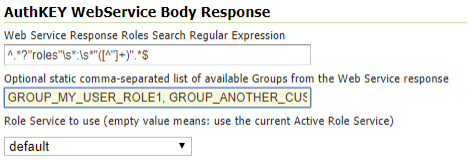
Now, this custom `UserGroup Service` maps dynamically `GeoServer User Role` to `GeoServer User Group` as follows:

```
ROLE_MY_USER_ROLE1                <> GROUP_MY_USER_ROLE1
ROLE_ANOTHER_CUSTOM_USER_ROLE     <> GROUP_ANOTHER_CUSTOM_USER_ROLE
ROLE_EXTERNAL_ROLE_X              <> GROUP_EXTERNAL_ROLE_X
```

In order to be able to assign any `GeoServer User Group` to other internal `GeoServer`



`User Roles`, since those are known only at runtime, the `UserGroup Service` allows us to **statically** specify the `GeoServer User Groups` the Web Service can use; this possible by setting the `Optional static comma-separated list of available Groups from the Web Service response` option:



Once this is correctly configured, it will be possible to edit and assign `GeoServer User Roles` to the Groups by using the standard way

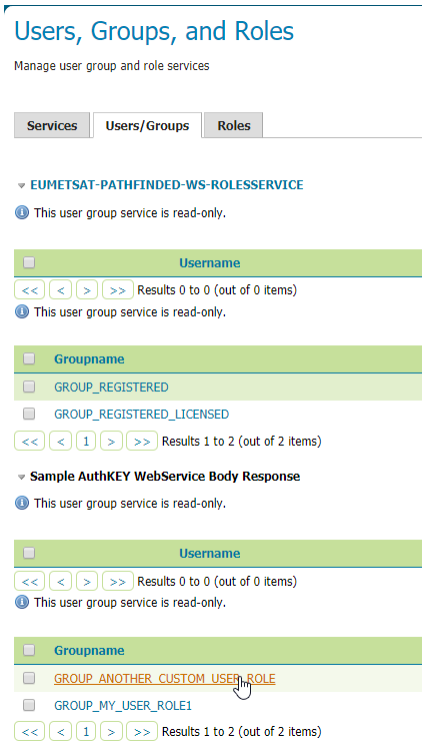
2. *Role Service to use*

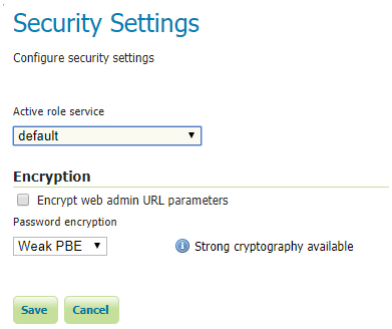
   By default, if no `Role Service` specified, the `UserGroup Service` will use the `GeoServer Active Role Service` to resolve `GeoServer User Roles` from `GeoServer User Groups` - as specified above -

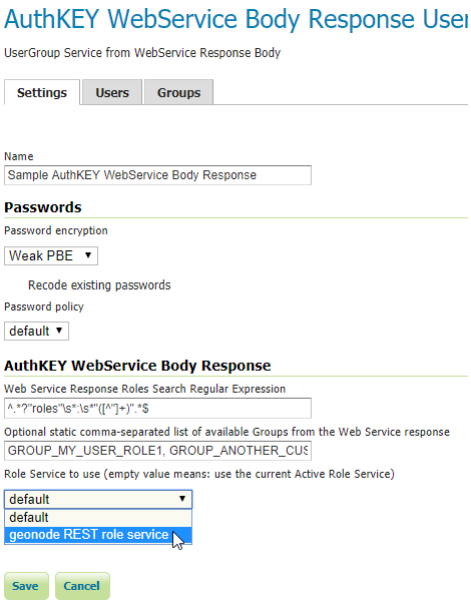
   It is possible to define a `Custom Role Service` to use instead, to resole `GeoServer User Roles`; this is possible simply by selecting the `Role Service` to use from the `Role Service to use` option

16.1.3 Configuration

Configuration can be done using the administrator GUI. There is a new type of authentication filter named **authkey** offering the following options.


Users, Groups, and Roles
Manage user group and role services
Services
Users/Groups
Roles
EUMETSAT-PATHFINDED-WS-ROLESSERVICE
This user group service is read-only.
Username
<< < > >> Results 0 to 0 (out of 0 items)
This user group service is read-only.
Groupname
GROUP_REGISTERED
GROUP_REGISTERED_LICENSED
<< < 1 > >> Results 1 to 2 (out of 2 items)
Sample AuthKEY WebService Body Response
This user group service is read-only.
Username
<< < > >> Results 0 to 0 (out of 0 items)
This user group service is read-only.
Groupname
GROUP_ANOTHER_CUSTOM_USER_ROLE
GROUP_MY_USER_ROLE1
<< < 1 > >> Results 1 to 2 (out of 2 items)


Security Settings
Configure security settings
Active role service
default
Encryption
Encrypt web admin URL parameters
Password encryption
Weak PBE
Strong cryptography available
Save
Cancel



1. URL parameter name. This the name of URL parameter used in client HTTP requests. Default is `authkey`.
2. Key Provider. GeoSever offers the providers described above.
3. User/group service to be used.

Some of the key providers can require additional configuration parameter. These will appear under the Key Provider combobox when one of those is selected.

After configuring the filter it is necessary to put this filter on the authentication filter chain(s).

Note: The administrator GUI for this filter has button **Synchronize**. Clicking on this button saves the current configuration and triggers a synchronize. If users are added/removed from the backing user/group service, the synchronize logic should be triggered.

16.1.4 Provider pluggability

With some Java programming it is possible to programmatically create and register a new key to user name mapper that works under a different logic. For example, you could have daily tokens, token generators and the like.

In order to provide your custom mapper you have to implement the `org.geoserver.security.AuthenticationKeyMapper` interface and then register said bean in the Spring application context. Alternatively it is possible to subclass from `org.geoserver.security.AbstractAuthenticationKeyMapper`. A mapper (key provider) has to implement

```
/**
 *
 * Maps a unique authentication key to a user name. Since user names are
 * unique within a {@link GeoServerUserGroupService} an individual mapper
 * is needed for each service offering this feature.
 *
 * @author Andrea Aime - GeoSolution
```

```
 */
public interface AuthenticationKeyMapper extends BeanNameAware {

    /**
     * Maps the key provided in the request to the {@link GeoServerUser} object
     * of the corresponding user, or returns null
     * if no corresponding user is found
     *
     * Returns <code>null</code> if the user is disabled
     *
     * @param key
     * @return
     */
    GeoServerUser getUser(String key) throws IOException;

    /**
     * Assures that each user in the corresponding {@link GeoServerUserGroupService}␣
↪has
     * an authentication key.
     *
     * returns the number of added authentication keys
     *
     * @throws IOException
     */
    int synchronize() throws IOException;

    /**
     * Returns <code>true</code> it the mapper can deal with read only u
     * user/group services
     *
     * @return
     */
    boolean supportsReadOnlyUserGroupService();

    String getBeanName();

    void setUserGroupServiceName(String serviceName);
    String getUserGroupServiceName();

    public GeoServerSecurityManager getSecurityManager();
    public void setSecurityManager(GeoServerSecurityManager securityManager);
```

}

The mapper would have to be registered in the Spring application context in a `applicationContext.xml` file in the root of your jar. Example for an implementation named `com.mycompany.security.SuperpowersMapper`:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN" "http://www.springframework.org/dtd/
↪spring-beans.dtd">
<beans>
  <bean id="superpowersMapper" class="com.mycompany.security.SuperpowersMapper"/>
</beans>
```

At this point you can drop the `authkey` jar along with your custom mapper jar and use it in the administrator GUI of the authentication key filter.

16.2 Authentication with OAuth2

This tutorial introduces GeoServer OAuth2 support and walks through the process of setting up authentication aganist an OAuth2 provider. It is recommended that the *Authentication chain* section be read before proceeding.

16.2.1 OAuth2 Protocol and GeoServer OAuth2 core module

This module allows GeoServer to authenticate against the OAuth2 Protocol.

In order to let the module work, it's mandatory to setup and configure both `oauth2` and `oauth2-xxxx-extension`.

The first one contains the necessary dependencies of the OAuth2 core module. This module contains the implementation of the GeoServer security filter, the base classes for the OAuth2 Token services and the GeoServer GUI panel.

Since in almost all cases the only thing different between OAuth2 Providers are the endpoint URIs and the client connection information (not only the keys - public and secret - but also the user profile representations), in order to allow GeoServer connecting to a specific OAuth2 provider it is sufficient to install the OAuth2 Core module plugin (and correctly configure the parameters through the GeoServer GUI - see next section for the details) and the concrete implementation of the OAuth2 REST token template and resource details.

Currently this module is shipped with a sample extension for Google OAuth2 Provider. This is a particular case since the Google JWT response is not standard and therefore we had to define and inject also a `GoogleUserAuthenticationConverter` taking the Google REST response against a valid `access_token` and converting it to a OAuth2 standard one.

Other than this the most interesting part is the implementation of the base class `GeoServerOAuth2SecurityConfiguration`.

The latter contains the Google implementation of the `OAuth2RestTemplate`.

In the next section we will see how to install and configure the OAuth2 security filter on GeoServer authenticating against Google OAuth2 Provider.

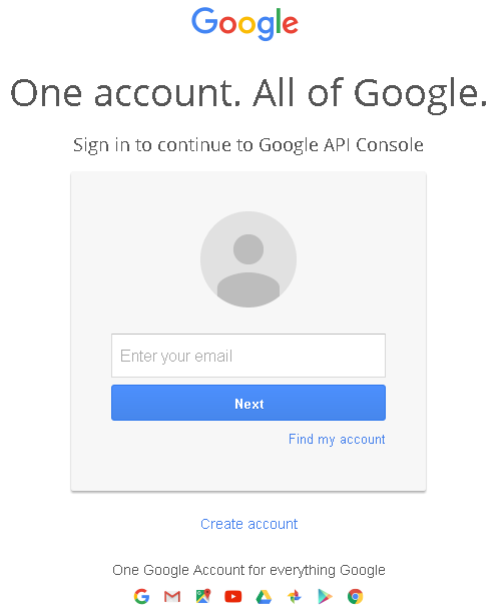
16.2.2 Configure the Google authentication provider

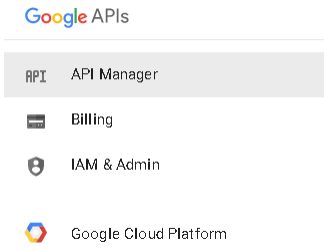
The first thing to do is to configure the OAuth2 Provider and obtain `Client ID` and `Client Sectet` keys.

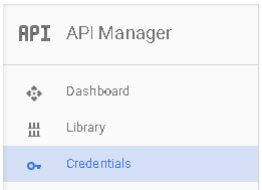
1. Obtain OAuth 2.0 credentials from the Google API Console.

   Visit the Google API Console to obtain OAuth 2.0 credentials such as a client ID and client secret that are known to both Google and your application. The set of values varies based on what type of application you are building. For example, a JavaScript application does not require a secret, but a web server application does.

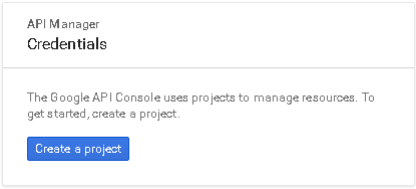
   - Login with a valid Google Account
   - Click on `API Manager`
   - Click on `Credentials`


Google
One account. All of Google.
Sign in to continue to Google API Console
Enter your email
Next
Find my account
Create account
One Google Account for everything Google

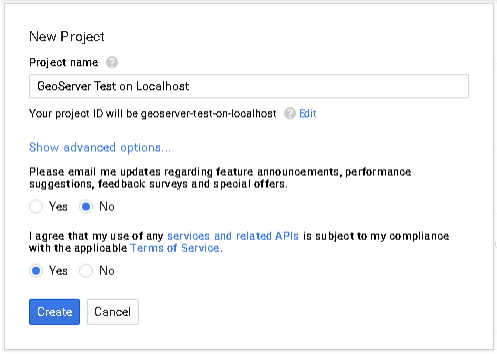

Google APIs
API Manager
Billing
IAM & Admin
Google Cloud Platform


API Manager
Dashboard
Library
Credentials

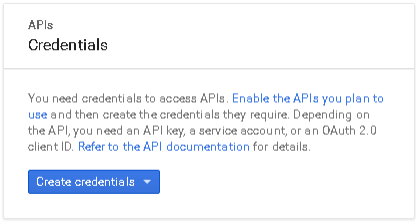
Note: The first time you land here, Google will ask to create at least one project



For the purpose of this tutorial we will create a sample project. You are free to create other projects or update existing ones through the Google API Console later.



If no `Credentials` are present, you will be asked to create new one.



2. Select an existing (or create a new one) `OAuth Client ID`

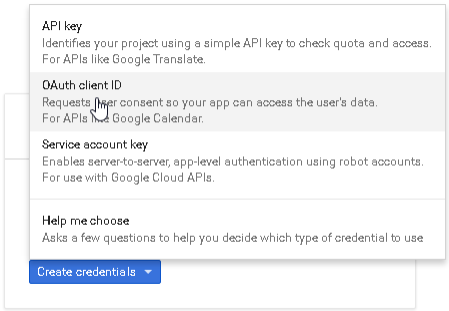
   Click on the `Client credentials` context menu as shown in the figure below.

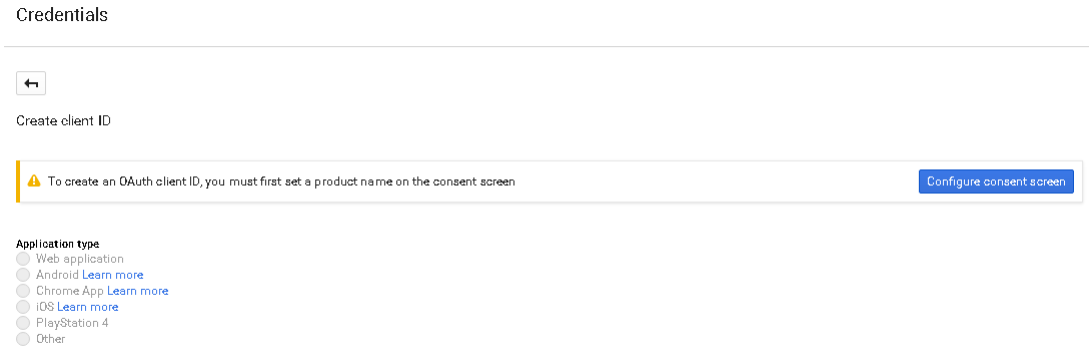
3. Configure a new `Web application`

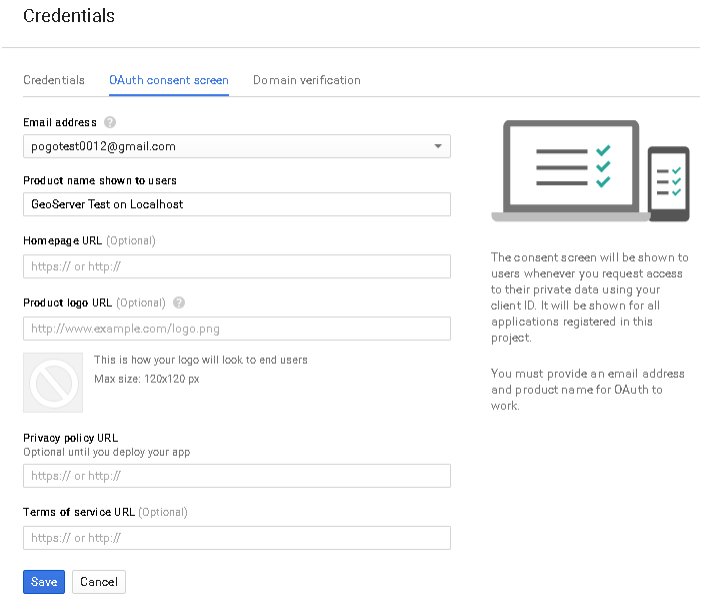
   - If it is the first time you create a `OAuth Client ID`, you will be asked to create a new `consent screen`

   - Customize the `consent screen`

   Warning: This step is mandatory only if it's the first time you are defining a `Web application` on a new project.


API key
Identifies your project using a simple API key to check quota and access.
For APIs like Google Translate.
OAuth client ID
Requests user consent so your app can access the user's data.
For APIs like Google Calendar.
Service account key
Enables server-to-server, app-level authentication using robot accounts.
For use with Google Cloud APIs.
Help me choose
Asks a few questions to help you decide which type of credential to use
Create credentials

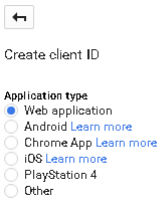

Credentials
Create client ID
To create an OAuth client ID, you must first set a product name on the consent screen
Configure consent screen
Application type
Web application
Android Learn more
Chrome App Learn more
iOS Learn more
PlayStation 4
Other


Credentials
Credentials
OAuth consent screen
Domain verification
Email address
pogotest0012@gmail.com
Product name shown to users
GeoServer Test on Localhost
Homepage URL (Optional)
https:// or http://
Product logo URL (Optional)
http://www.example.com/logo.png
This is how your logo will look to end users
Max size: 120x120 px
Privacy policy URL
Optional until you deploy your app
https:// or http://
Terms of service URL (Optional)
https:// or http://
Save
Cancel
The consent screen will be shown to users whenever you request access to their private data using your client ID. It will be shown for all applications registered in this project.
You must provide an email address and product name for OAuth to work.

Note: It can be edited and updated also later (see last point of this section below)

- Select `Application type` -> `Web application`

Warning: This step is mandatory only if it's the first time you are defining a `Web application` on a new project.

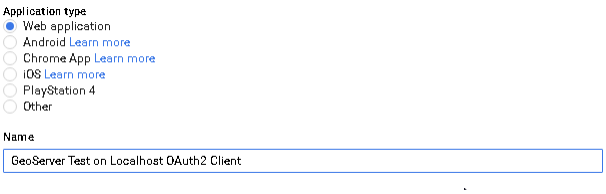


- Add a `Name` and the `Authorized redirect URIs` like shown here below.

Note: This sample creates a client working on the default local URL `http://localhost:8080/geoserver`. Of course this will work only on a local instance and can't be used for a production system.

However it is possible to add as many `Authorized redirect URIs` you need to a new `Web application`.

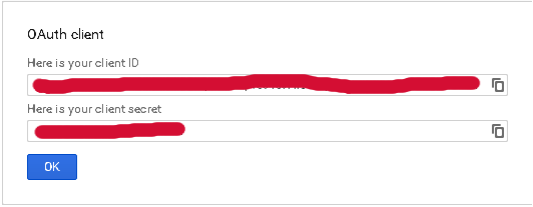
It is also possible create many `Client credentials` with customised `consent screen` and `Web application`, depending on your specific needs. Every public GeoServer instance (or cluster of GeoServer belonging to a specific project) should have it's own specific `Client credentials`.



Note: Always add two entries for each URI. One without the ending / and another one with it.

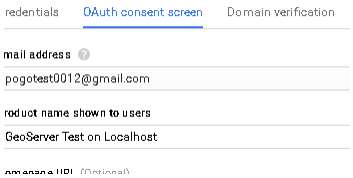
4. Click on `Create` and take note of the `Client ID` and the `Client Sectet`.

   At the end of the procedure Google will show-up a small dialog box with the `Client ID` and the `Client Sectet`. Those info can be always accessed and updated from the Google API Console



5. Optionally customize the `OAuth consent screen`.

   At any time it is possible to update and customize the `OAuth consent screen`. You can put here your logo, app name, ToS and so on.



16.2.3 Configure the GeoServer OAuth2 filter

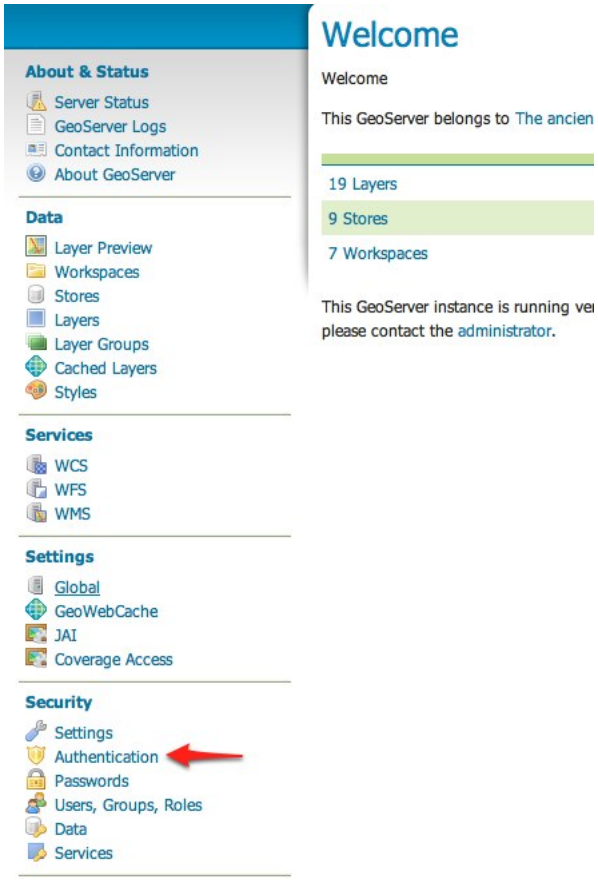
1. Start GeoServer and login to the web admin interface as the `admin` user.
2. Click the `Authentication` link located under the `Security` section of the navigation sidebar.
3. Scroll down to the `Authentication Filters` panel and click the `Add new` link.
4. Click the `OAuth2` link.
5. Fill in the fields of the settings form as follows:

   The default values provided with the plugin are valid for the Google OAuth2 Provider and are the following one:

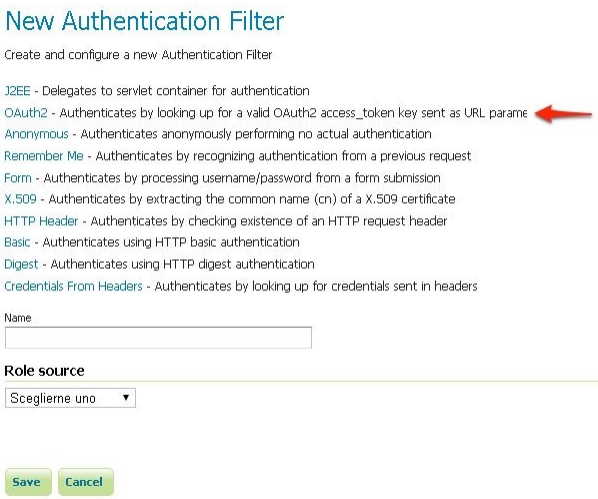
```
"Enable Redirect Authentication EntryPoint" = False
"Access Token URI" = https://accounts.google.com/o/oauth2/token
"User Authorization URI" = https://accounts.google.com/o/oauth2/auth
"Redirect URI" = http://localhost:8080/geoserver
"Check Token Endpoint URL" = https://www.googleapis.com/oauth2/v1/
↪tokeninfo
"Logout URI" = https://accounts.google.com/logout
"Scopes" = https://www.googleapis.com/auth/userinfo.email,https://www.
↪googleapis.com/auth/userinfo.profile
```

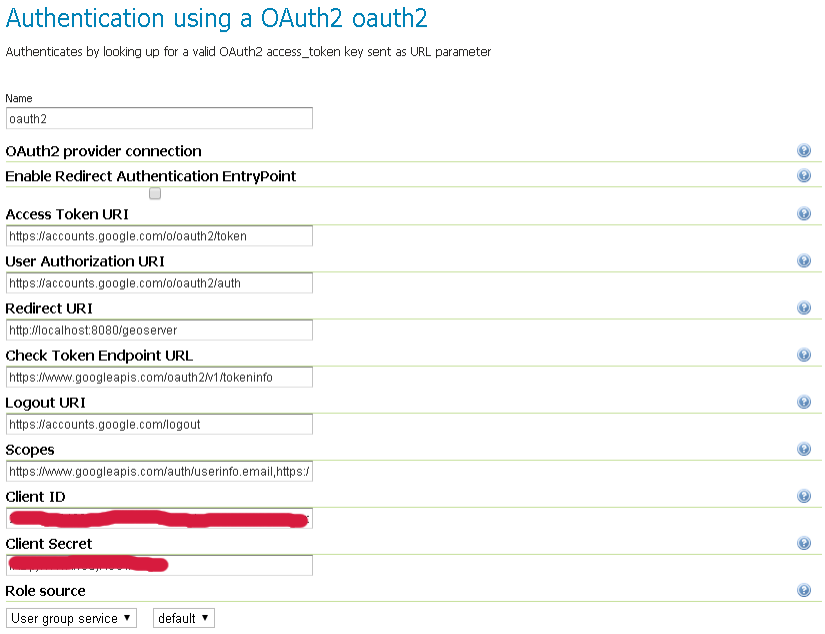
   Note:

   (a) `Client ID` and `Client Secret` are the ones Google provided


Welcome
About & Status
Server Status
GeoServer Logs
Contact Information
About GeoServer
Data
Layer Preview
Workspaces
Stores
Layers
Layer Groups
Cached Layers
Styles
Services
WCS
WFS
WMS
Settings
Global
GeoWebCache
JAI
Coverage Access
Security
Settings
Authentication
Passwords
Users, Groups, Roles
Data
Services
Welcome
This GeoServer belongs to The ancien
19 Layers
9 Stores
7 Workspaces
This GeoServer instance is running ver
please contact the administrator.


Authentication
Authentication providers and settings
Logout settings
Redirect URL after logout (empty,absolute or relative to context root)
/web/
SSL settings
SSL Port (default is 443)
443
Authentication Filters
Add new
Remove selected


New Authentication Filter
Create and configure a new Authentication Filter
J2EE - Delegates to servlet container for authentication
OAuth2 - Authenticates by looking up for a valid OAuth2 access_token key sent as URL parame
Anonymous - Authenticates anonymously performing no actual authentication
Remember Me - Authenticates by recognizing authentication from a previous request
Form - Authenticates by processing username/password from a form submission
X.509 - Authenticates by extracting the common name (cn) of a X.509 certificate
HTTP Header - Authenticates by checking existence of an HTTP request header
Basic - Authenticates using HTTP basic authentication
Digest - Authenticates using HTTP digest authentication
Credentials From Headers - Authenticates by looking up for credentials sent in headers
Name
Role source
Sceglierne uno
Save
Cancel


Authentication using a OAuth2 oauth2
Authenticates by looking up for a valid OAuth2 access_token key sent as URL parameter
Name
oauth2
OAuth2 provider connection
Enable Redirect Authentication EntryPoint
Access Token URI
https://accounts.google.com/o/oauth2/token
User Authorization URI
https://accounts.google.com/o/oauth2/auth
Redirect URI
http://localhost:8080/geoserver
Check Token Endpoint URL
https://www.googleapis.com/oauth2/v1/tokeninfo
Logout URI
https://accounts.google.com/logout
Scopes
https://www.googleapis.com/auth/userinfo.email,https:/
Client ID
Client Secret
Role source
User group service
default

(b) Choose a `Role Service` able to recognize user emails as IDs. By default a connected user will have `ROLE_USER` role

---

Warning: Few workds on **Enable Redirect Authentication EntryPoint** option

This option allows you to decide whether or not *force* automatic redirection to OAuth2 Access Token URI or not for authentication.

What does that means?

- *Enable Redirect Authentication EntryPoint* = True

  If not already authenticated (or no valid **Access Token** is provided in the query string), this option will **force** a redirection to the OAuth2 Provider Login page.

  This may cause unwanted behavior since it will override every other esplicit login method like `form`. In other words if the filter is applied for instance to the `web` endpoint, it won't be possible to access to the GeoServer Admin GUI using the standard login method via browser.

- *Enable Redirect Authentication EntryPoint* = False

  In order to avoid the above issue, by disabling this option you will be **forced** to use an esplicit Authentication Endpoint to login via the OAuth2 Provider login page.

  If not already authenticated (or no valid **Access Token** is provided in the query string), you **must** authenticate through the following URLs:

  (a) *GeoServer OAuth2 Authorization Endpoint*; `http://<host:port>/geoserver/j_spring_outh2_login`

  (b) *OAuth2 Provider Esplicit User Authorization Endpoint*; this must be adapted for your specific OAuth2 Provider, the protocol stated that it should be

  ```
  https://<USER_AUTHORIZATION_URI>?scope=<SCOPES>&
  ↪response_type=code&redirect_uri=<REDIRECT_URI>&
  ↪client_id=<CLIENT_ID>
  ```

  For google OAuth2 Provider is:

  ```
  https://accounts.google.com/o/oauth2/auth?scope
  ↪%3Dhttps://www.googleapis.com/auth/userinfo.
  ↪email%2Bhttps://www.googleapis.com/auth/
  ↪userinfo.profile%26response_type%3Dcode
  ↪%26redirect_uri%3D<REDIRECT_URI>%26client_id%3D
  ↪<CLIENT_ID>
  ```

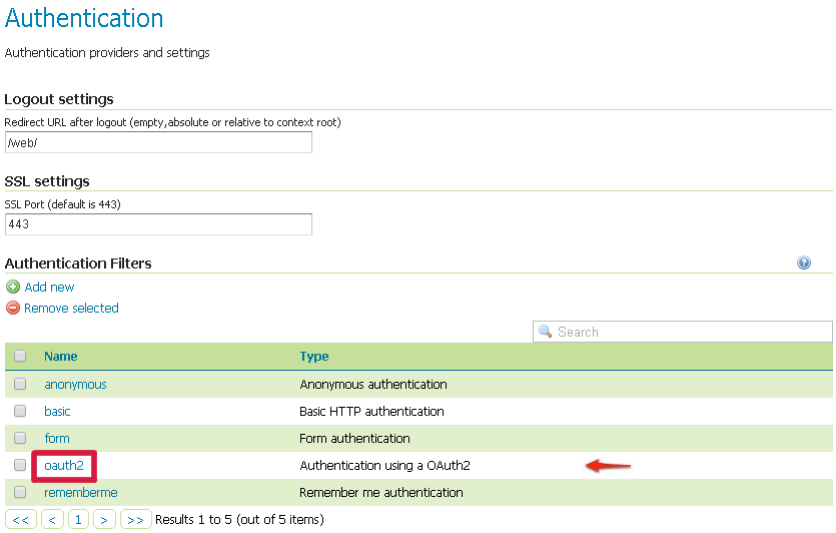
6. Update the filter chains by adding the new OAuth2 filter.

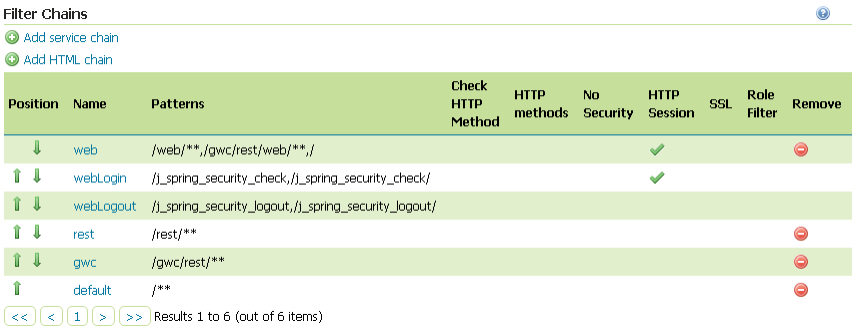
   Once everything has been configured you should be able to see the new `oauth2` filter available among the `Authentication Filters` list

   Through this it will be always possible to modify / update the filter options, or create more of them.

   The next step is to add the filter to the `Filter Chains` you want to protect with OAuth2 also

7. Select the OAuth2 Filter for each filter chain you want to protect with OAuth2.


Authentication
Authentication providers and settings
Logout settings
Redirect URL after logout (empty,absolute or relative to context root)
/web/
SSL settings
SSL Port (default is 443)
443
Authentication Filters
Add new
Remove selected
Search
Name
Type
anonymous
Anonymous authentication
basic
Basic HTTP authentication
form
Form authentication
oauth2
Authentication using a OAuth2
rememberme
Remember me authentication
<< < 1 > >> Results 1 to 5 (out of 5 items)


Filter Chains
Add service chain
Add HTML chain
Position
Name
Patterns
Check HTTP Method
HTTP methods
No Security
HTTP Session
SSL
Role Filter
Remove
web
/web/**,/gwc/rest/web/**,/
webLogin
/j_spring_security_check,/j_spring_security_check/
webLogout
/j_spring_security_logout,/j_spring_security_logout/
rest
/rest/**
gwc
/gwc/rest/**
default
/**
<< < 1 > >> Results 1 to 6 (out of 6 items)

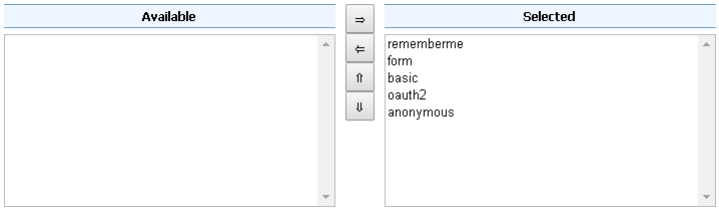
If you need to protect **all** the GeoServer services and the GeoServer Admin GUI too with OAuth2, you need to add the `oauth2` filter to all the following chains

- `web`
- `rest`
- `gwc`
- `default`

The order of the authentication filters depends basically on which method you would like GeoServer *try first*.

Note: During the authtentication process, the authentication filters of a `Filter Chain` are executed serially until one succeed (for more details please see the section *Authentication chain*)

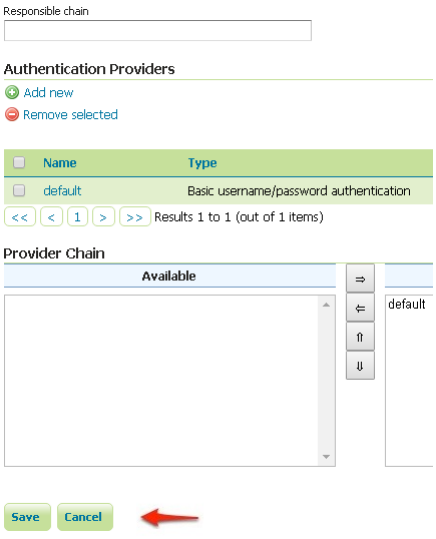
Warning: If *Enable Redirect Authentication EntryPoint* = **True** for OAuth2 Filter, the `web` chain won't be able to login through the `form` method.



Note: Remember that the `anonymous` filter must be always the last one.

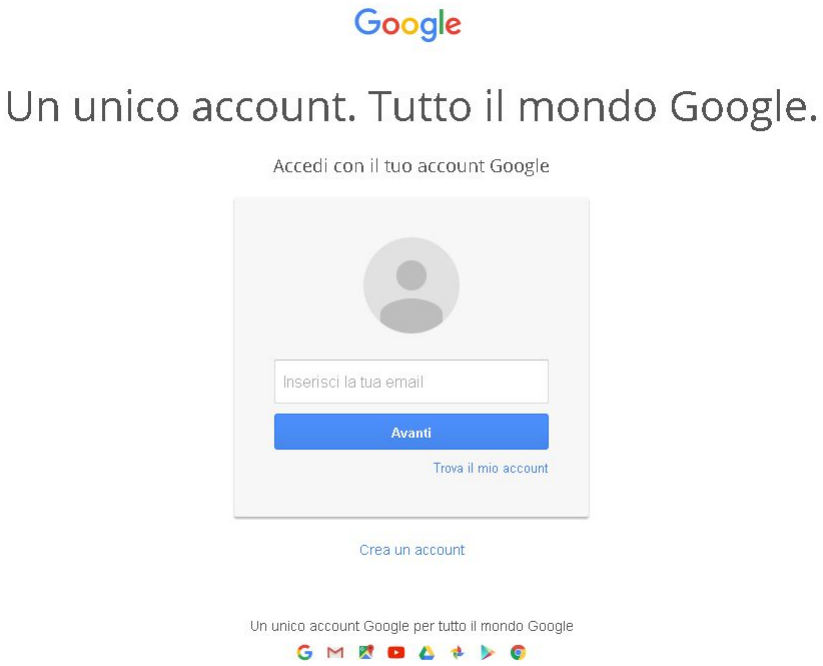
---

8. Save.



It's now possible to test the authentication:

1. Navigate to the GeoServer home page and log out of the admin account.
2. Try to login again, you should be able now to see the external Google login form.



16.2.4 OpenID connect authentication

The OpenID connect authentication is working in a way quite similar to Google (and GitHub) authentications, the only difference is that the authentication page cannot propose default values for the various endpoints, which have to be configured manually.

In case the web login will not be used, the "client ID" and "client secret" are not actually needed, and can be filled with two made up values (the validation just checks they are present, but they will be used only in the "authorisation flow", but not when doing OGC requests where the client is supposed to have autonomously retrieved a valid bearer token).

Warning: The oauth2-openid-connect does not implement the full protocol and has been tested against a single server, more development and testing is needed before it can be consumed by a wider audience. Pull requests to improve the module are welcomed.

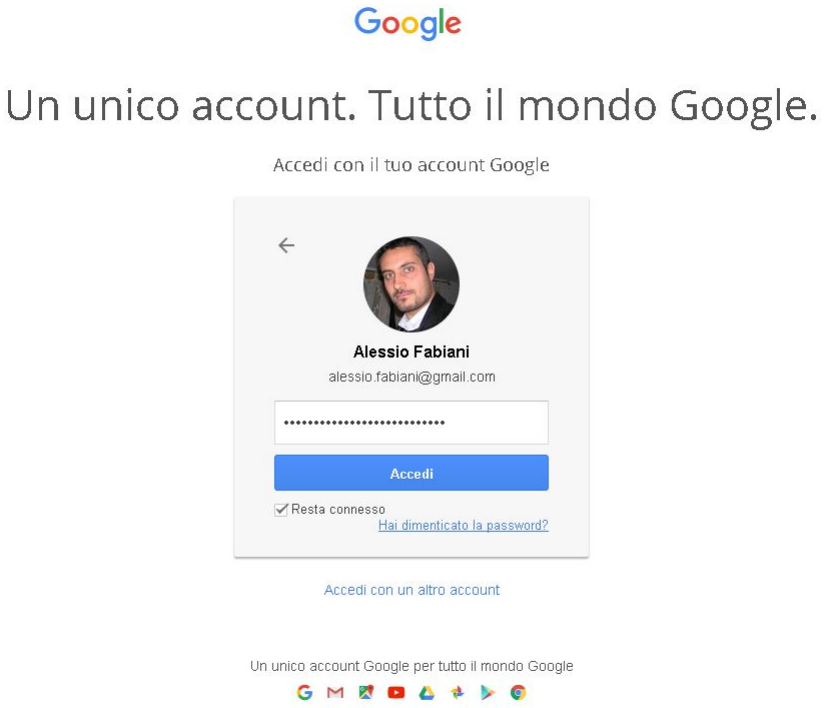
16.2.5 SSL Trusted Certificates

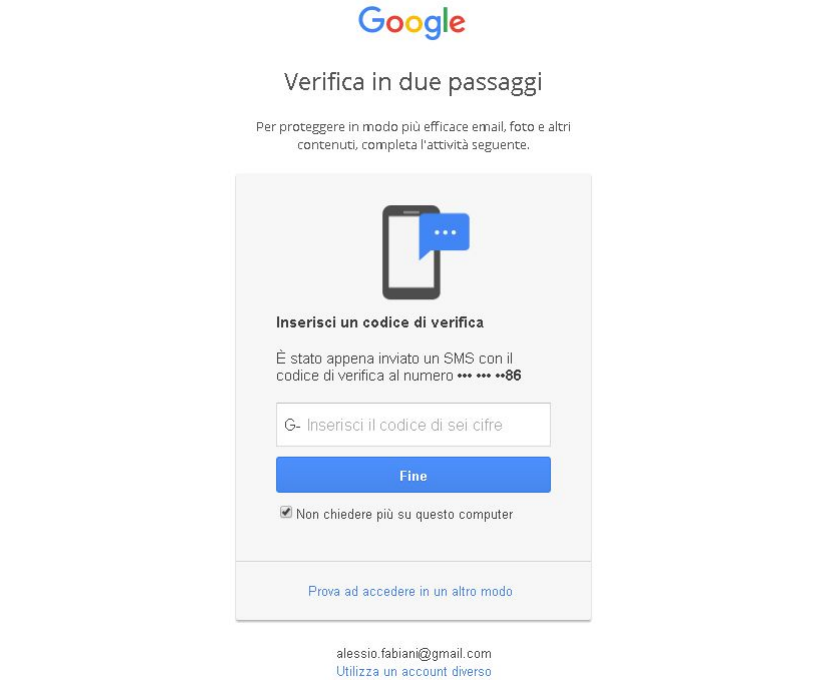
When using a custom `Keystore` or trying to access a non-trusted or self-signed SSL-protected OAuth2 Provider from a non-SSH connection, you will need to add the certificated to the JVM `Keystore`.

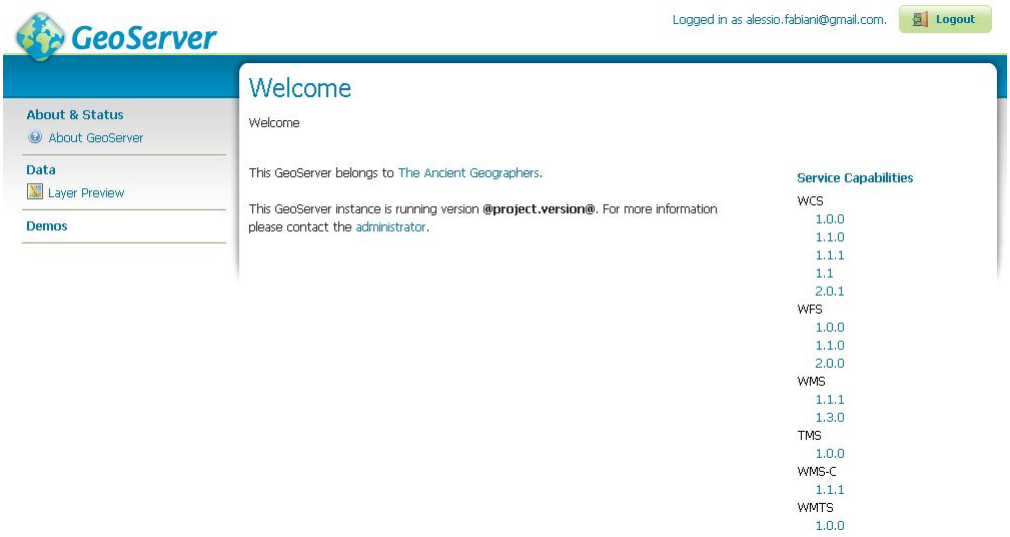
In order to do this you can follow the next steps:

In this example we are going to

1. Retrieve SSL Certificates from Google domains:


Google
Un unico account. Tutto il mondo Google.
Accedi con il tuo account Google
Alessio Fabiani
alessio.fabiani@gmail.com
Accedi
Resta connesso
Hai dimenticato la password?
Accedi con un altro account
Un unico account Google per tutto il mondo Google


Google
Verifica in due passaggi
Per proteggere in modo più efficace email, foto e altri contenuti, completa l'attività seguente.
Inserisci un codice di verifica
È stato appena inviato un SMS con il codice di verifica al numero ••• ••• ••86
G- Inserisci il codice di sei cifre
Fine
Non chiedere più su questo computer
Prova ad accedere in un altro modo
alessio.fabiani@gmail.com
Utilizza un account diverso



"Access Token URI" = https://accounts.google.com/o/oauth2/token therefore we need to trust `https://accounts.google.com` or (`accounts.google.com:443`) "Check Token Endpoint URL" = https://www.googleapis.com/oauth2/v1/tokeninfo therefore we need to trust `https://www.googleapis.com` or (`www.googleapis.com:443`)

Note: You will need to get and trust certificated from every different HTTPS URL used on OAuth2 Endpoints.

---

2. Store SSL Certificates on local hard-disk
3. Add SSL Certificates to the Java Keystore
4. Enable the JVM to check for SSL Certificates from the Keystore

1. Retrieve the SSL Certificates from Google domains

   Use the `openssl` command in order to dump the certificate

   For `https://accounts.google.com`

   ```
   openssl s_client -connect accounts.google.com:443
   ```

   And for `https://www.googleapis.com`

   ```
   openssl s_client -connect www.googleapis.com:443
   ```

```
Loading 'screen' into random state - done
CONNECTED(00000188)
depth=3 C = US, O = Equifax, OU = Equifax Secure Certificate Authority
verify return:1
depth=2 C = US, O = GeoTrust Inc., CN = GeoTrust Global CA
verify return:1
depth=1 C = US, O = Google Inc, CN = Google Internet Authority G2
verify return:1
depth=0 C = US, ST = California, L = Mountain View, O = Google Inc, CN = accounts.google.com
verify return:1
---
Certificate chain
 0 s:/C=US/ST=California/L=Mountain View/O=Google Inc/CN=accounts.google.com
   i:/C=US/O=Google Inc/CN=Google Internet Authority G2
 1 s:/C=US/O=Google Inc/CN=Google Internet Authority G2
   i:/C=US/O=GeoTrust Inc./CN=GeoTrust Global CA
 2 s:/C=US/O=GeoTrust Inc./CN=GeoTrust Global CA
   i:/C=US/O=Equifax/OU=Equifax Secure Certificate Authority
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIEoTCCA4mgAwIBAgIIeEP870cN1RQwDQYJKoZIhvcNAQELBQAwSTELMAkGA1UE
BhMCVVMxEzARBgNVBAoTCkdvb2dsZSBJbmMxJTAjBgNVBAMTHEdvb2dsZSBJbnRl
cm5ldCBBdXRob3JpdHkgRzIwHhcNMTYxMDE5MTcxNjU3WhcNMTcwMTExMTcxMzAw
WjBtMQswCQYDVQQGEwJVUzETMBEGA1UECAwKQ2FsaWZvcm5pYTEWMBQGA1UEBwwN
TW91bnRhaW4gVmlldzETMBEGA1UECgwKR29vZ2xlIEluYzEcMBoGA1UEAwwTYWNj
b3VudHMuZ29vZ2xlLmNvbTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEB
AK71x4DYCS7NcMqyp8UAq7067Wb7LLhkgE5QJcUVdvFfLrXmG3PAIcfiLMvdIK+F
RODn5d79I18qgn9OXgB/xyFgx381MR/HoglavblCTfgbiotvhAMsxyY/h55efbcn
i2GGOXCWVcx119xPOnWFvd7aUaOLSutMeEZ7T4Hqj3oOWFYVJq+dQNGdkGwqtUap
bosQFuyWg+rKINM6Opvy6ay8fRU/4JeKNVcaZUuIu12bPchBzIXiKtLkMJSIYzRz
ySjJmFLm++/ipMQ549xeyviB4pSz44athBuIgENhgE8Xb7eTS5TPuWFjJ913nhdT
IDsalPMX2PpsinA1IF8Z+kECAwEAAaOCAWcwggFjMB0GA1UdJQQWMBQGCCsGAQUF
BwMBBggrBgEFBQcDAjA1BgNVHREELjAsghNhY2NvdW50cy5nb29nbGUuY29tghUq
```

```
Loading 'screen' into random state - done
CONNECTED(00000194)
depth=3 C = US, O = Equifax, OU = Equifax Secure Certificate Authority
verify return:1
depth=2 C = US, O = GeoTrust Inc., CN = GeoTrust Global CA
verify return:1
depth=1 C = US, O = Google Inc, CN = Google Internet Authority G2
verify return:1
depth=0 C = US, ST = California, L = Mountain View, O = Google Inc, CN = *.googleapis.com
verify return:1
---
Certificate chain
 0 s:/C=US/ST=California/L=Mountain View/O=Google Inc/CN=*.googleapis.com
   i:/C=US/O=Google Inc/CN=Google Internet Authority G2
 1 s:/C=US/O=Google Inc/CN=Google Internet Authority G2
   i:/C=US/O=GeoTrust Inc./CN=GeoTrust Global CA
 2 s:/C=US/O=GeoTrust Inc./CN=GeoTrust Global CA
   i:/C=US/O=Equifax/OU=Equifax Secure Certificate Authority
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIE3TCCA8WgAwIBAgIIIZQW3eg4WCgwDQYJKoZIhvcNAQELBQAwSTELMAkGA1UE
BhMCVVMxEzARBgNVBAoTCkdvb2dsZSBJbmMxJTAjBgNVBAMTHEdvb2dsZSBJbnRl
cm5ldCBBdXRob3JpdHkgRzIwHhcNMTYxMDE5MTcxMzI5WhcNMTcwMTExMTcxMzAw
WjBqMQswCQYDVQQGEwJVUzETMBEGA1UECAwKQ2FsaWZvcm5pYTEWMBQGA1UEBwwN
TW91bnRhaW4gVmlldzETMBEGA1UECgwKR29vZ2xlIEluYzEZMBcGA1UEAwwQKi5n
b29nbGVhcGlzLmNvbTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAIWL
6Jh7kLkFW62UwwRuILPaKY+ajI2GU66IADIqJDfKObtxUCgyPweCnnJZYaIwqU8q
oG2TzTtam25xdOOxOFuU2qTodvMNSt54Li29VMVRLQuIlQrlIiXPa8CVXZ5rCwY1
sH7e7gJOUNwZZusj7HUxdOHrVDTtsrkyQOJcKlPzAzFazizsEFM4Pkwfgp1IcCav
yf8dPrE2gOp9nJl2G/2DrwLIep+CldKCRu9jU+Ss3iHLM8/KH6pag15UPw9FY8+F
qhRdE7xpR60nHnnqAMaDT6MtQhJO1/NBfJvORnOJHEd3c/VLmM7knB6WytMMy+DO
cP2ZuaFXGcLQ1Nc39tsCAwEAAaOCAaYwggGiMB0GA1UdJQQWMBQGCCsGAQUFBwMB
BggrBgEFBQcDAjBOBgNVHREERzBFghNhY2NvdW50cy5nb29nbGUuY29tghUqLmNsaWVu
dHM2Lmdvb2dsZS5jb22CGCouY2xvdWRlbmRwb2ludHNhcGlzLmNvbYIWY2xvdWRl
```

2. Store SSL Certificates on local hard-disk

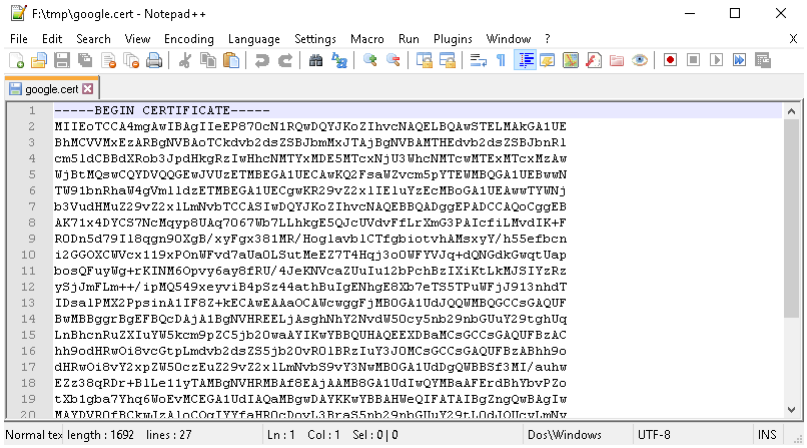
   Copy-and-paste the two sections `-BEGIN CERTIFICATE-`, `-END CERTIFICATE-` and save them into two different `.cert` files

   ---

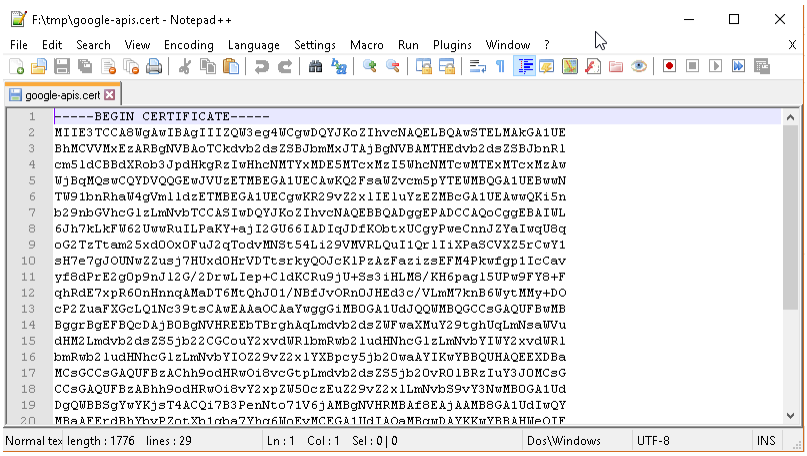
   Note: `.cert` file are plain text files containing the ASCII characters included on the `-BEGIN CERTIFICATE-`, `-END CERTIFICATE-` sections

   ---

   `google.cert` (or whatever name you want with `.cert` extension)



   `google-apis.cert` (or whatever name you want with `.cert` extension)



3. Add SSL Certificates to the Java Keystore

   You can use the Java command `keytool` like this

   `google.cert` (or whatever name you want with `.cert` extension)

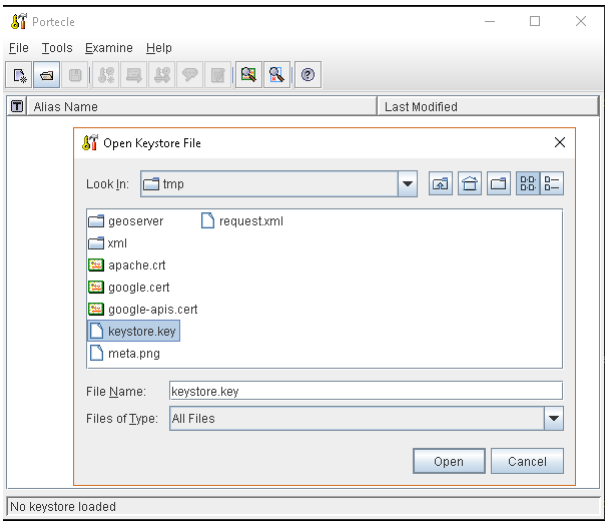
```
keytool -import -noprompt -trustcacerts -alias google -file␣
↪google.cert -keystore ${KEYSTOREFILE} -storepass $
↪{KEYSTOREPASS}
```

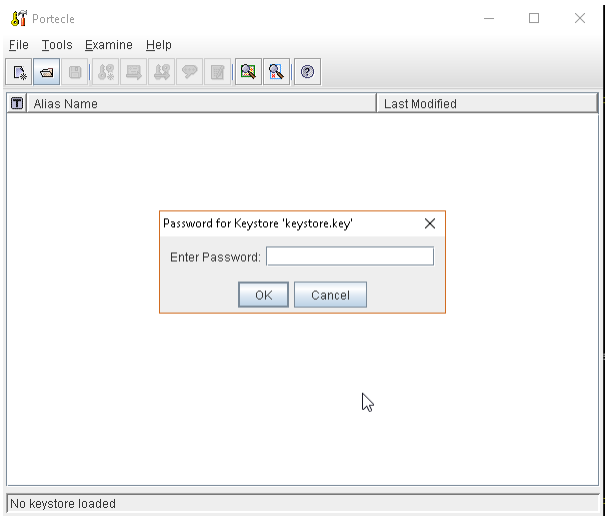
   `google-apis.cert` (or whatever name you want with `.cert` extension)

```
keytool -import -noprompt -trustcacerts -alias google-apis -
↪file google-apis.cert -keystore ${KEYSTOREFILE} -
↪storepass ${KEYSTOREPASS}
```

or, alternatively, you can use some graphic tool which helps you managing the SSL Certificates and Keystores, like Portecle

```
java -jar c:\apps\portecle-1.9\portecle.jar
```





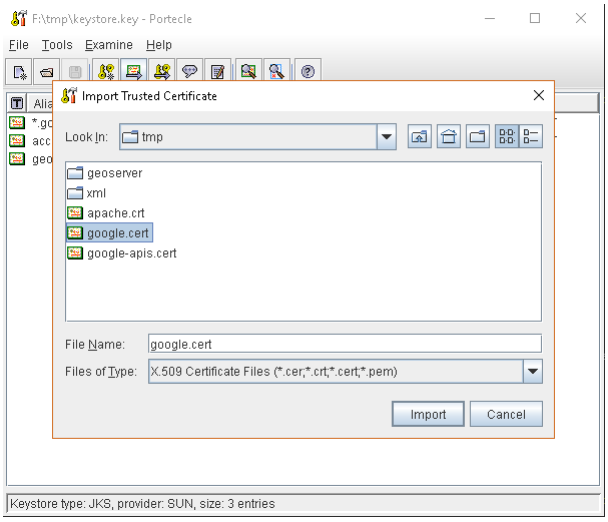
4. Enable the JVM to check for SSL Certificates from the Keystore

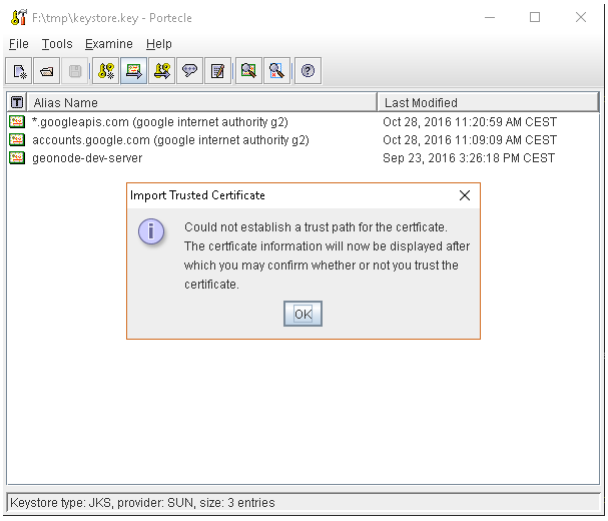
   In order to do this, you need to pass a `JAVA_OPTION` to your JVM:

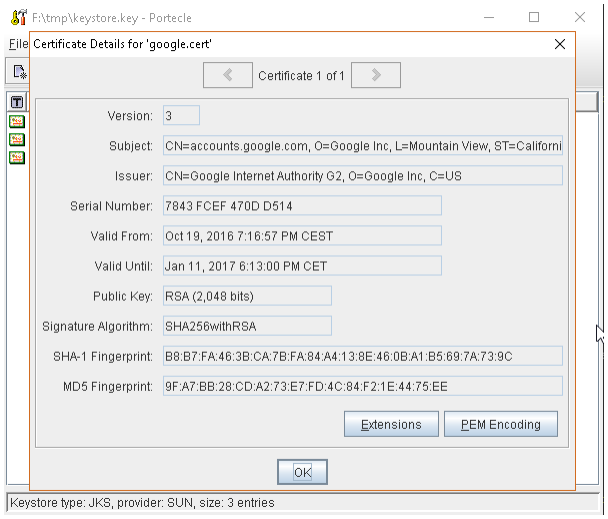
   ```
   -Djavax.net.ssl.trustStore=F:\tmp\keystore.key
   ```

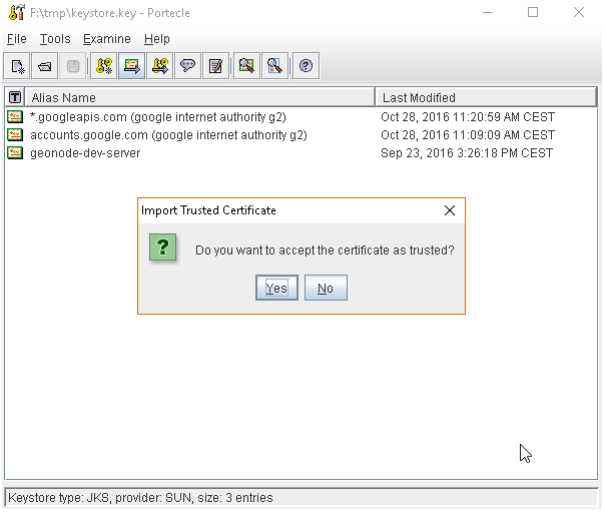
5. Restart your server

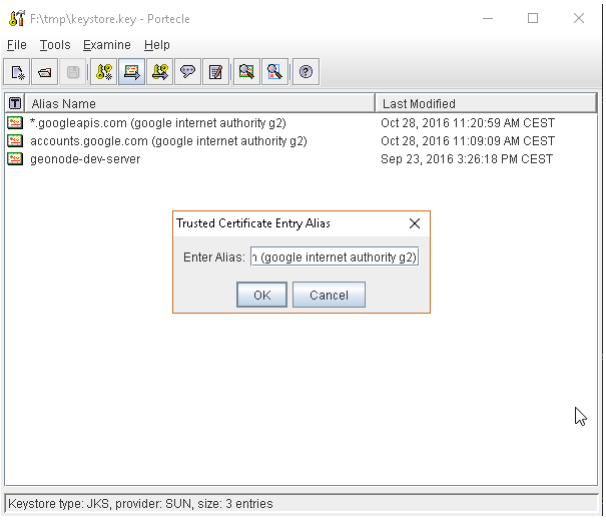
Note: Here below you can find a bash script which simplifies the Keystore SSL Certificates importing. Use it at your conveninece.

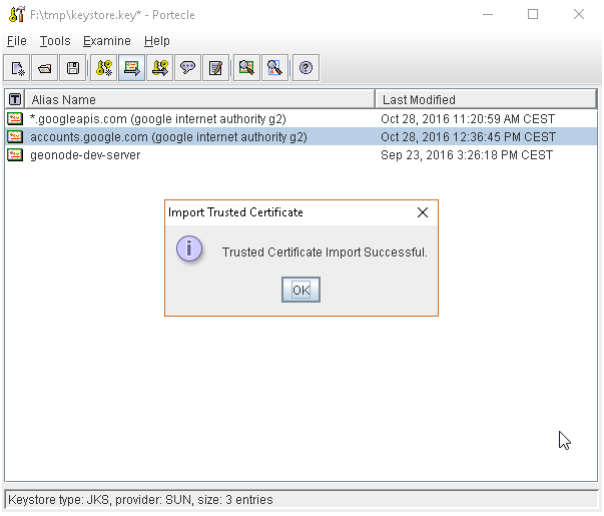

F:\tmp\keystore.key - Portecle
File Tools Examine Help
Import Trusted Certificate
Look In: tmp
geoserver
xml
apache.crt
google.cert
google-apis.cert
File Name: google.cert
Files of Type: X.509 Certificate Files (*.cer,*.crt,*.cert,*.pem)
Import
Cancel
Keystore type: JKS, provider: SUN, size: 3 entries

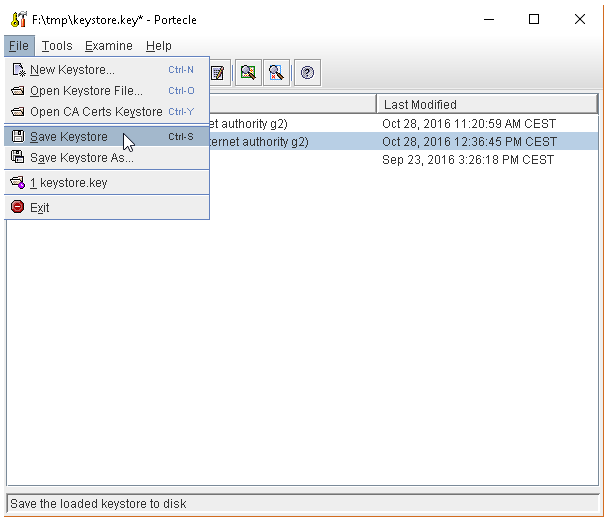

F:\tmp\keystore.key - Portecle
File Tools Examine Help
Alias Name
Last Modified
*.googleapis.com (google internet authority g2)
Oct 28, 2016 11:20:59 AM CEST
accounts.google.com (google internet authority g2)
Oct 28, 2016 11:09:09 AM CEST
geonode-dev-server
Sep 23, 2016 3:26:18 PM CEST
Import Trusted Certificate
Could not establish a trust path for the certficate.
The certficate information will now be displayed after which you may confirm whether or not you trust the certificate.
OK
Keystore type: JKS, provider: SUN, size: 3 entries


F:\tmp\keystore.key - Portecle
File
Certificate Details for 'google.cert'
Certificate 1 of 1
Version: 3
Subject: CN=accounts.google.com, O=Google Inc, L=Mountain View, ST=Californi
Issuer: CN=Google Internet Authority G2, O=Google Inc, C=US
Serial Number: 7843 FCEF 470D D514
Valid From: Oct 19, 2016 7:16:57 PM CEST
Valid Until: Jan 11, 2017 6:13:00 PM CET
Public Key: RSA (2,048 bits)
Signature Algorithm: SHA256withRSA
SHA-1 Fingerprint: B8:B7:FA:46:3B:CA:7B:FA:84:A4:13:8E:46:0B:A1:B5:69:7A:73:9C
MD5 Fingerprint: 9F:A7:BB:28:CD:A2:73:E7:FD:4C:84:F2:1E:44:75:EE
Extensions
PEM Encoding
OK
Keystore type: JKS, provider: SUN, size: 3 entries


F:\tmp\keystore.key - Portecle
File Tools Examine Help
Alias Name
Last Modified
*.googleapis.com (google internet authority g2)
Oct 28, 2016 11:20:59 AM CEST
accounts.google.com (google internet authority g2)
Oct 28, 2016 11:09:09 AM CEST
geonode-dev-server
Sep 23, 2016 3:26:18 PM CEST
Import Trusted Certificate
Do you want to accept the certificate as trusted?
Yes
No
Keystore type: JKS, provider: SUN, size: 3 entries


F:\tmp\keystore.key - Portecle
File Tools Examine Help
Alias Name
Last Modified
*.googleapis.com (google internet authority g2)
Oct 28, 2016 11:20:59 AM CEST
accounts.google.com (google internet authority g2)
Oct 28, 2016 11:09:09 AM CEST
geonode-dev-server
Sep 23, 2016 3:26:18 PM CEST
Trusted Certificate Entry Alias
Enter Alias:
n (google internet authority g2)
OK
Cancel
Keystore type: JKS, provider: SUN, size: 3 entries


F:\tmp\keystore.key* - Portecle
File Tools Examine Help
Alias Name
Last Modified
*.googleapis.com (google internet authority g2)
Oct 28, 2016 11:20:59 AM CEST
accounts.google.com (google internet authority g2)
Oct 28, 2016 12:36:45 PM CEST
geonode-dev-server
Sep 23, 2016 3:26:18 PM CEST
Import Trusted Certificate
Trusted Certificate Import Successful.
OK
Keystore type: JKS, provider: SUN, size: 3 entries




```
HOST=myhost.example.com
PORT=443
KEYSTOREFILE=dest_keystore
KEYSTOREPASS=changeme

# get the SSL certificate
openssl s_client -connect ${HOST}:${PORT} </dev/null \
        | sed -ne '/-BEGIN CERTIFICATE-/,/-END CERTIFICATE-/p' > ${HOST}.cert

# create a keystore and import certificate
keytool -import -noprompt -trustcacerts \
        -alias ${HOST} -file ${HOST}.cert \
        -keystore ${KEYSTOREFILE} -storepass ${KEYSTOREPASS}

# verify we've got it.
keytool -list -v -keystore ${KEYSTOREFILE} -storepass ${KEYSTOREPASS} -alias ${HOST}
```

16.3 Authentication with Keycloak

This tutorial introduces GeoServer Keycloak support and walks through the process of setting up authentication aganist an Keycloak provider. It is recommended that the *Authentication chain* section be read before proceeding.

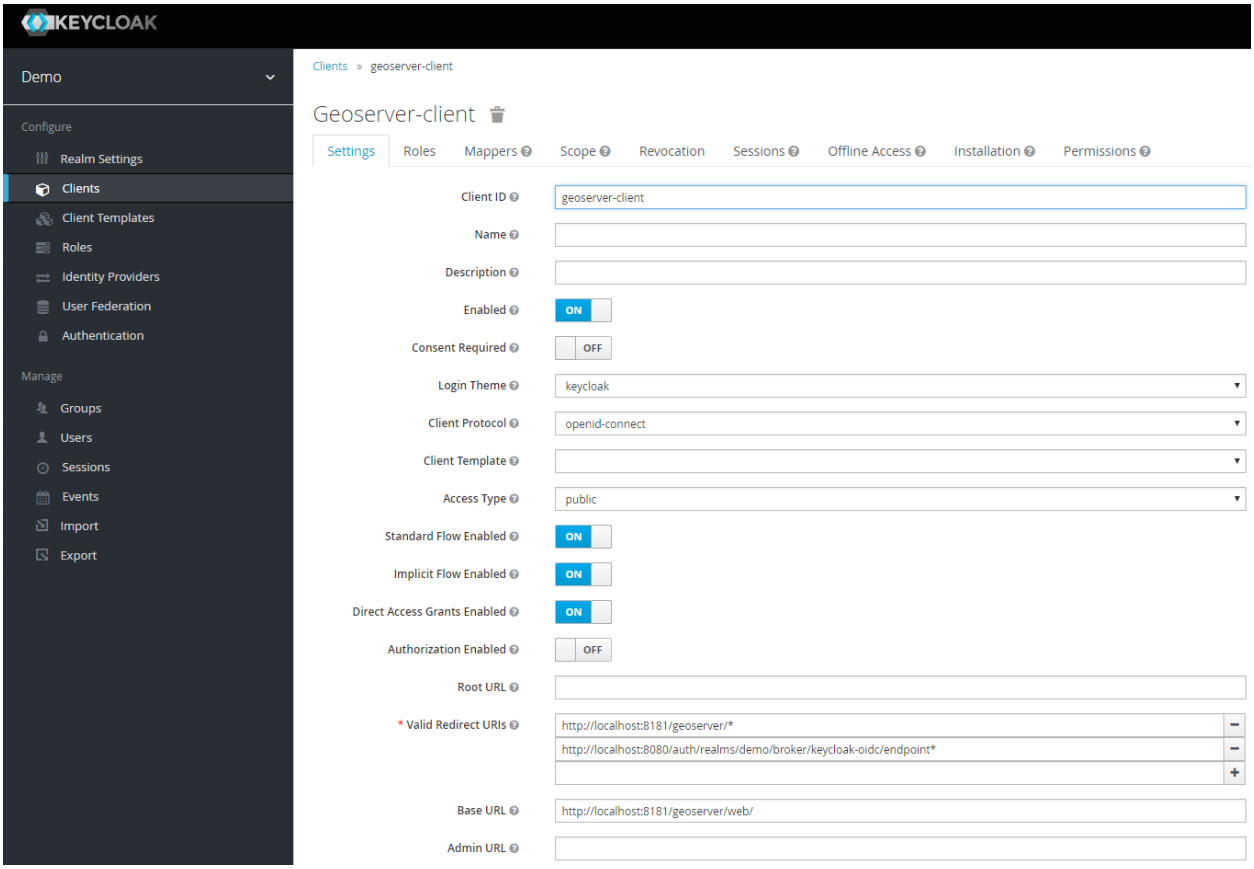
The GeoServer Keycloak-authn/authz plugin will allow you to use an instance of Keycloak to control access to resources within GeoServer.

16.3.1 Installtation Instructions

As the Keycloak Admin:

Note: In this example the Keycloak service runs on port *8080* while GeoServer runs on port *8181*

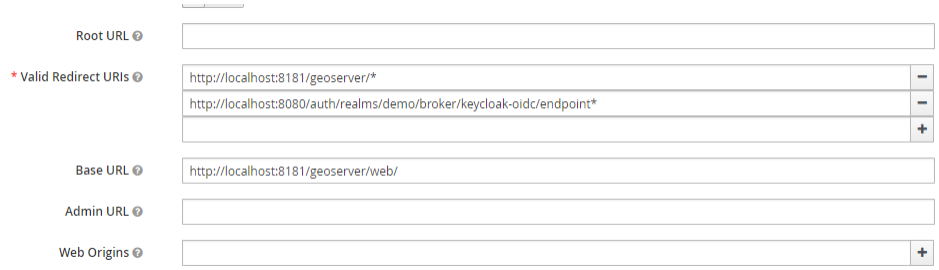
1. Create a new client for GeoServer named *geoserver-client*.



2. Make sure to add the base URL of GeoServer to the list of acceptable redirect paths, and add also the Keycloak OIDC endpoint base URI.

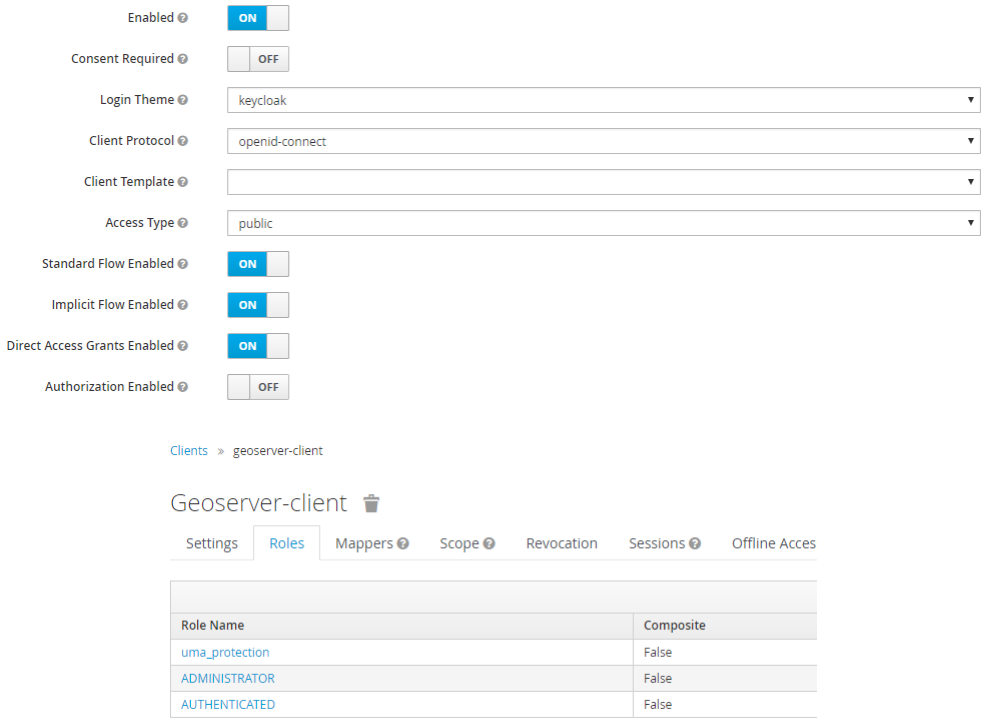
   eg:

   - http://localhost:8181/geoserver*
   - http://localhost:8080/auth/realms/demo/broker/keycloak-oidc/endpoint*

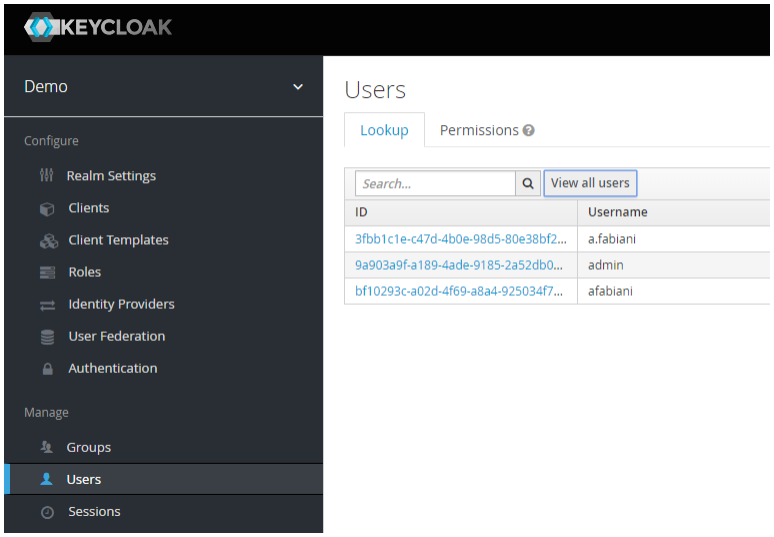


2. Set the *access-type* of client as appropriate. If your GeoServer instance is depending on another service for authentication (eg: NGINX auth plugin) then you should probably select *bearer-only*. Otherwise, you should probably select *public*.

3. Add the the *ADMINISTRATOR* and *AUTHENTICATED* client-role to the *geoserver-client* in Keycloak.

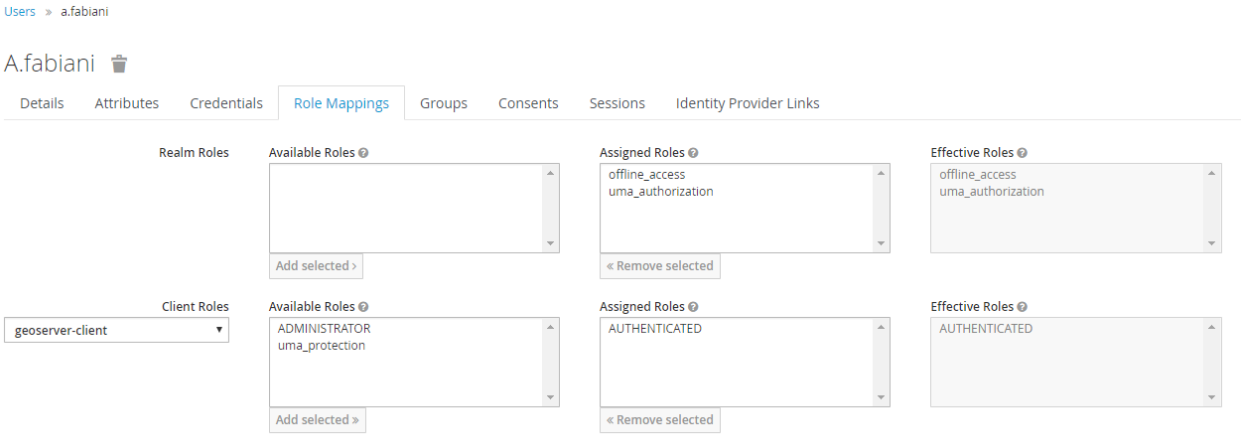


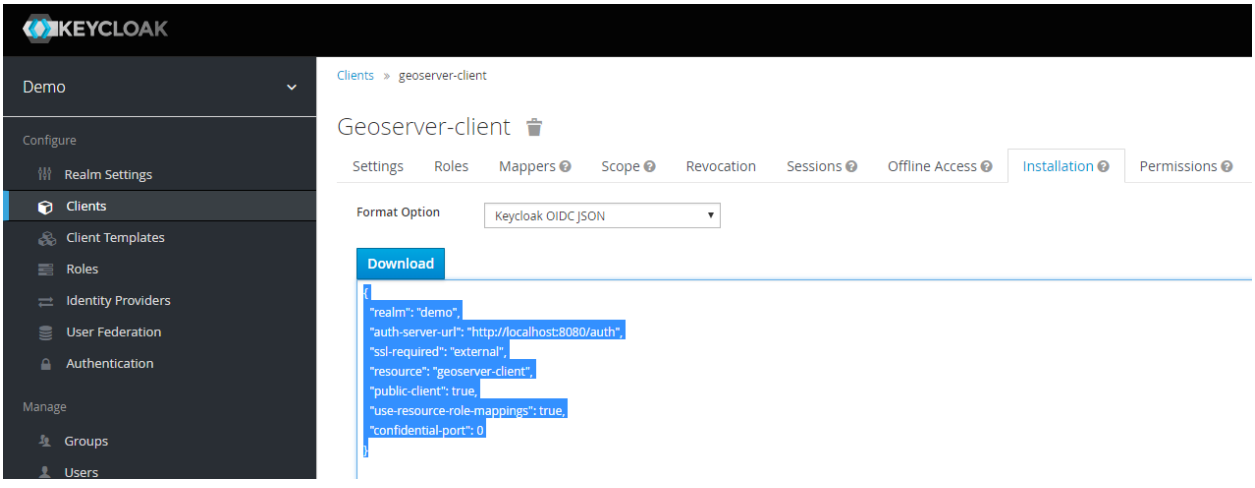
In this phase you will need to map GeoServer Roles to the *geoserver-client* ones in Keycloak.



Use the *AUTHENTICATED* one for generic users. Assign this role *ADMINISTRATOR* to the users/groups who should have administrative access to GeoServer.

4. Obtain the installation-configuration for the *geoserver-client* in JSON, and provide this to the GeoServer Admin for the next steps.


Users » a.fabiani
A.fabiani
Details
Attributes
Credentials
Role Mappings
Groups
Consents
Sessions
Identity Provider Links
Realm Roles
Available Roles
Assigned Roles
offline_access
uma_authorization
Effective Roles
offline_access
uma_authorization
Add selected ›
« Remove selected
Client Roles
geoserver-client
Available Roles
ADMINISTRATOR
uma_protection
Assigned Roles
AUTHENTICATED
Effective Roles
AUTHENTICATED
Add selected »
« Remove selected


KEYCLOAK
Demo
Configure
Realm Settings
Clients
Client Templates
Roles
Identity Providers
User Federation
Authentication
Manage
Groups
Users
Clients » geoserver-client
Geoserver-client
Settings
Roles
Mappers
Scope
Revocation
Sessions
Offline Access
Installation
Permissions
Format Option
Keycloak OIDC JSON
Download
{
"realm": "demo",
"auth-server-url": "http://localhost:8080/auth",
"ssl-required": "external",
"resource": "geoserver-client",
"public-client": true,
"use-resource-role-mappings": true,
"confidential-port": 0
}

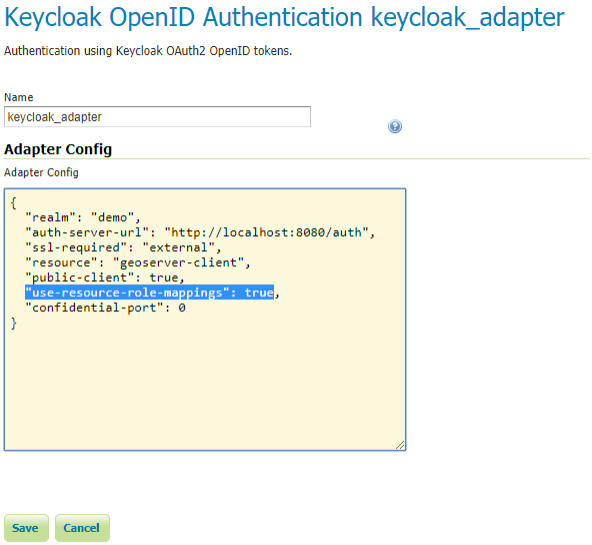
As the GeoServer Admin:

Note: In this example the Keycloak service runs on port *8080* while GeoServer runs on port *8181*

1. Under the Authentication UI, add a new *authentication-filter*. Select *Keycloak* from the list of provided options, and name your new filter *keycloak_adapter*. Paste the installation-configuration from the Keycloak-server in the text area provided.

   If not present, be sure to add the following options before clicking *Save*:

   ```
   "use-resource-role-mappings": true
   ```

2. Add the *keycloak_adapter* to the *web filter-chain* if you want to protect the Admin GUI, as an instance.

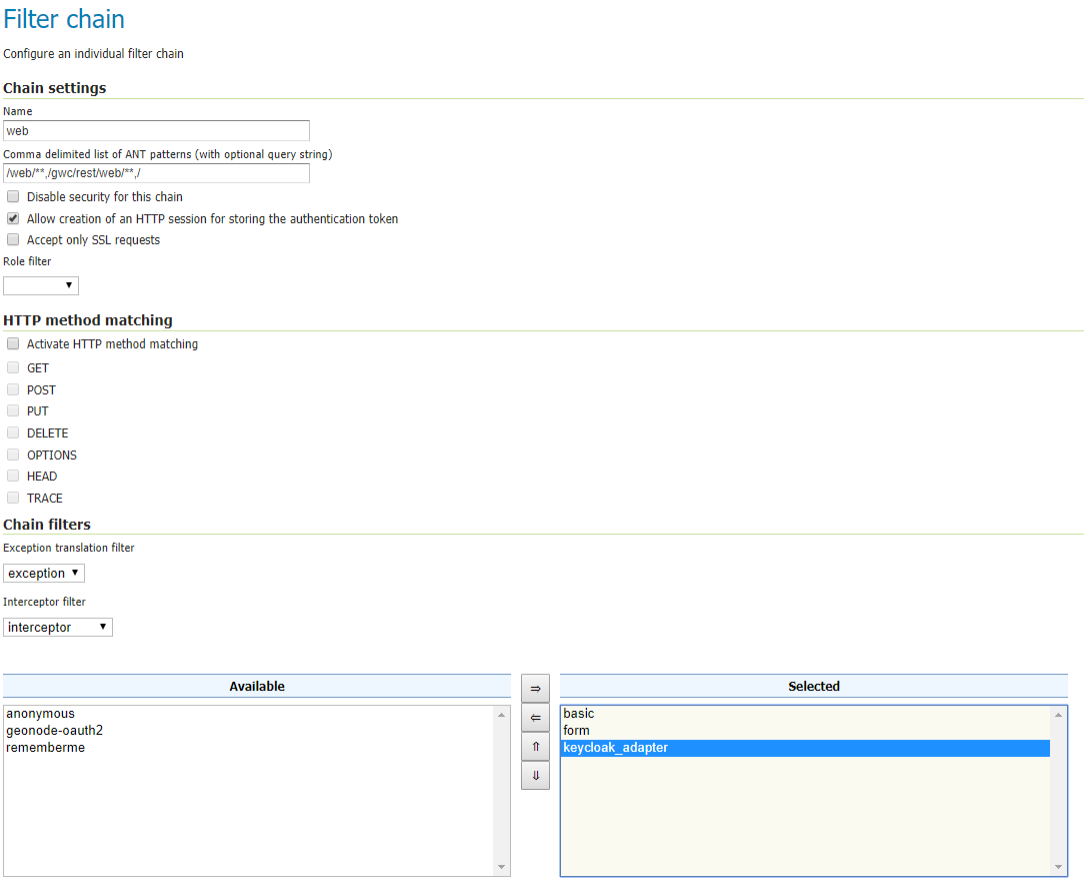
3. Check your work so far by navigating to the GeoServer UI. You should be redirected to the Keycloak *login-page*, and after logging-in you should be redirected back to the actual GUI page.

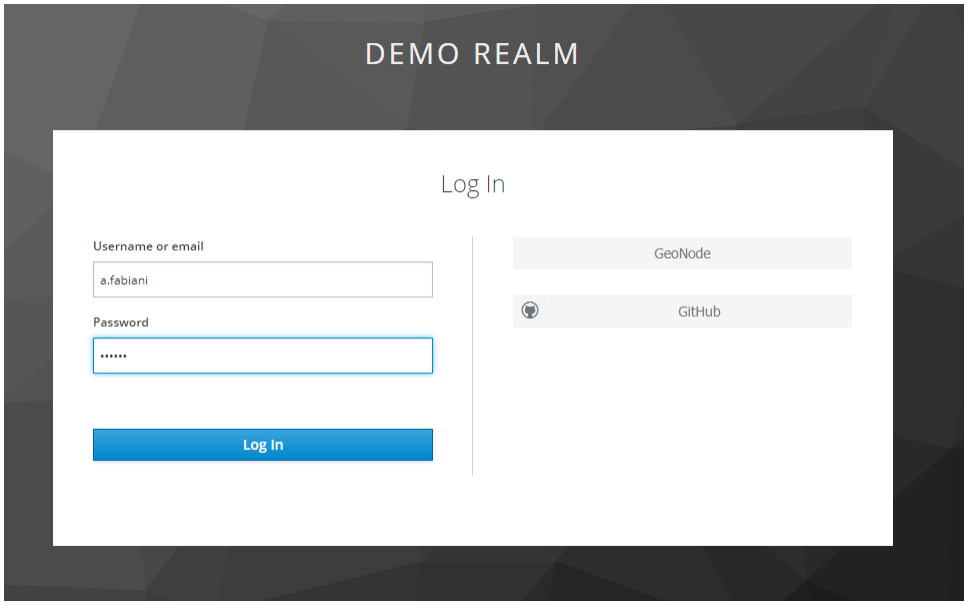
   You should verify that the message *logged in as <USERNAME>* is posted in the top right corner before continuing.

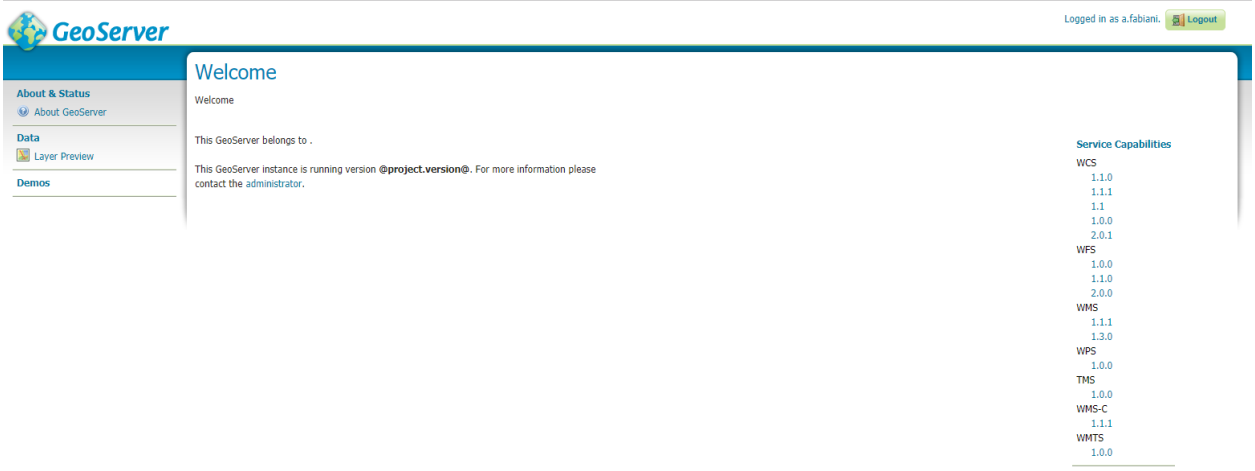
16.4 DDS/BIL(World Wind Data Formats) Extension

This output module allows GeoServer to output imagery and terrain in formats understood by NASA World Wind. The mime-types supported are:

1. Direct Draw Surface (DDS) - image/dds. This format allows efficient loading of textures to the GPU and takes the task off the WorldWind client CPU in converting downloaded PNG, JPEG or TIFF tiles. The DDS compression is done using DXT3 with help from the worldwind library on server side.


Filter chain
Configure an individual filter chain
Chain settings
Name
web
Comma delimited list of ANT patterns (with optional query string)
/web/**,/gwc/rest/web/**,/
Disable security for this chain
Allow creation of an HTTP session for storing the authentication token
Accept only SSL requests
Role filter
HTTP method matching
Activate HTTP method matching
GET
POST
PUT
DELETE
OPTIONS
HEAD
TRACE
Chain filters
Exception translation filter
exception
Interceptor filter
interceptor
Available
anonymous
geonode-oauth2
rememberme
Selected
basic
form
keycloak_adapter


DEMO REALM
Log In
Username or email
a.fabiani
Password
Log In
GeoNode
GitHub



2. Binary Interleaved by Line(BIL) - image/bil. This is actually a very simple raw binary format produced using the RAW Image Writer. The supplied GridCoverage2D undergoes appropriate subsampling, reprojection and bit-depth conversion. The output can be requested as 16bit Int or 32bit Float.

16.4.1 Installing the DDS/BIL extension

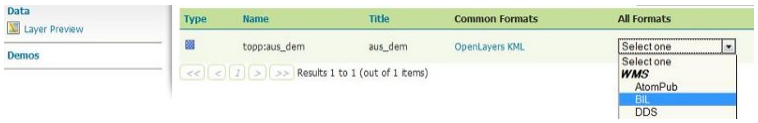
1. Download the DDS/BIL extension from the nightly GeoServer community module builds. A prebuilt version for GeoServer 2.0.x can be found on Jira - GEOS-3586.

> **Warning:** Make sure to match the version of the extension to the version of the GeoServer instance!

2. Extract the contents of the archive into the `WEB-INF/lib` directory of the GeoServer installation.

16.4.2 Checking if the extension is enabled

Once the extension is installed, the provided mime-types should appear in the layer preview dropbox as shown:



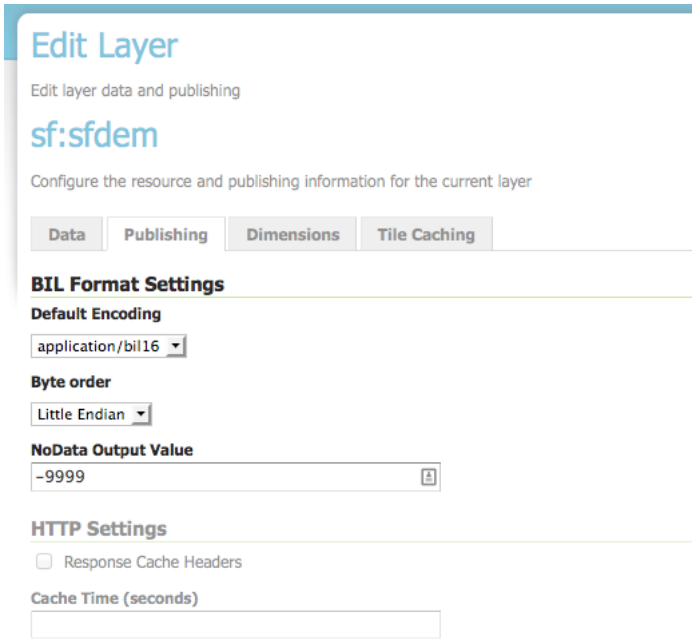
The mime-types will also be listed in the `GetCapabilities` document:

```
<Format>image/bil</Format>
<Format>image/dds</Format>
```

16.4.3 Configuring the BIL format

For a client application to use a BIL layer, it must know the data encoding of the BIL file (e.g. 16-bit integer, 32-bit floating point, etc), the byte order of the data, and the value that indicates missing data. BIL files do

not contain this metadata, so it may be necessary to configure the server to produce BIL files in the format that a client application expects.



The BIL output format can be configured for each layer in the Publishing tab of the layer configuration. The plugin supports the following options:

Option	Description
`Default encoding`	The data encoding to use if the request does not specify an encoding. For example, application/bil does not specify the response encoding, while application/bil16 does specify an encoding. Default: use same encoding as layer source files.
`Byte order`	Byte order of the response. Default: network byte order (big endian).
`No Data value`	The value that indicates missing data. If this option is set, missing data values will be recoded to this value. Default: no data translation.

For compatibility with the default behavior of NASA World Wind, use these settings:

- Default encoding: application/bil16
- Byte order: Little endian
- No data: -9999

16.4.4 Configuring World Wind to access Imagery/Terrain from GeoServer

Please refer to the WorldWind Forums for instructions on how to setup World Wind to work with layers published via GeoServer. For image layers(DDS) the user need to create a WMSTiledImageLayer either via XML configuration or programmatically. For terrain layers (BIL) the equivalent class is WMSBasicElevationModel.

16.5 Scripting

The GeoServer scripting extension allows users to extend GeoServer dynamically by writing scripts in languages other than Java.

16.5.1 Installing the Scripting Extension

Python

Currently, the only scripting language that is distributed as a package for download is Python. This extension is a community extension, in that it is not included with the list of extensions on the standard GeoServer download page. Instead, the community extensions are built each night on the nightly build server.

To access the Python scripting extension:

1. Navigate to the nightly build server.
2. Click the folder that contains the correct branch of GeoServer for your version (for example: for 2.2.2, click on *2.2.x*):
3. Click *community-latest*. This folder contains the most recently built community extensions for the branch.
4. Download the file that contains the string "python". For example: `geoserver-2.2-SNAPSHOT-python-plugin.zip`.
5. Extract the contents of the archive into the `/WEB-INF/lib/` directory of GeoServer. For example, if GeoServer was installed at `/opt/geoserver-2.2.2/`, extract the archive contents in `/opt/geoserver-2.1.0/webapps/geoserver/WEB-INF/lib/`.
6. Restart GeoServer.

Upon a successful install a new directory named `scripts` will be created inside the data directory.

16.5.2 Supported Languages

Support for the following scripting languages is available:

- Python
- JavaScript
- Groovy
- Beanshell
- Ruby

Adding support for additional languages is relatively straight forward. The requirements for adding a new language are:

1. The language has an implementation that runs on the Java virtual machine
2. The language runtime provides a JSR-223 compliant script engine

GeoScript

GeoScript is a project that adds scripting capabilities to the GeoTools library. It can be viewed as bindings for GeoTools in various other languages that are supposed on the JVM. It is the equivalent of the various language bindings that GDAL and OGR provide.

Currently GeoScript is available for the following languages:

- Python
- JavaScript
- Groovy

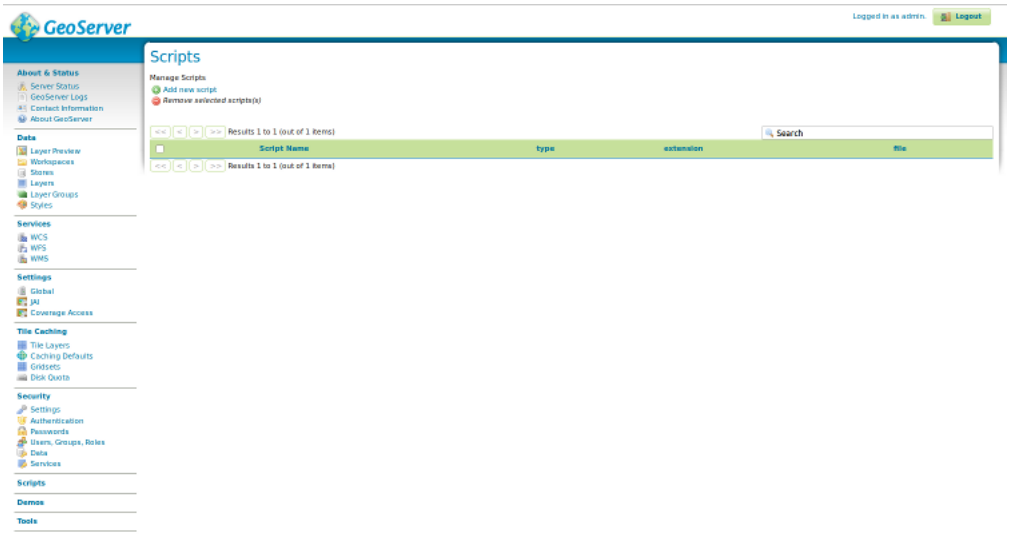
The associated GeoServer scripting extension for these languages come with GeoScript for that language enabled. This means that when writing scripts one has access to the GeoScript modules and packages like they would any other standard library package.

Those languages that don't have a GeoScript implementation can still implement the same functionality that GeoScript provides but must do it against the GeoTools api directly. The downside being that usually the GeoTools api is much more verbose than the GeoScript equivalent. But the upside is that going straight against the GeoTools api is usually more efficient.

Therefore GeoScript can be viewed purely as a convenience for script writers.

16.5.3 Scripting Web User Interface

After successful installation you should see a Scripts menu item:



For adding new script click "Add new script":

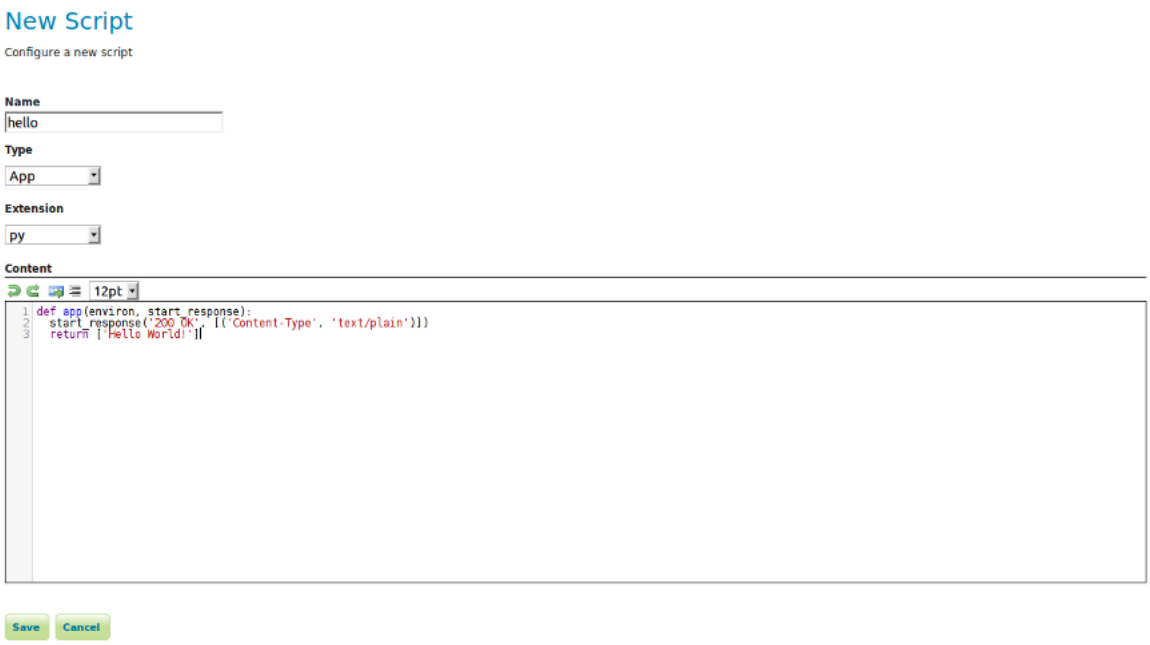
Input parameters:

Name— The name of the script file (should be added without extension);

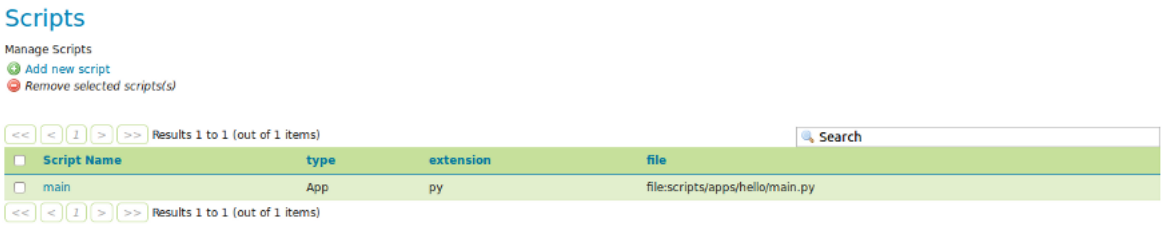
Type— Script extension point, see *Scripting Extension Overview* and *Scripting Hooks*;

Extension— The file extension for the scripting language of your choice;

Content— Your script, for examples, see *Scripting Hooks* and *Scripting Reference*;



After saving a script you should see the created file path relative to data directory root.



16.5.4 Scripting Extension Overview

The scripting extension provides a number of extension points called "hooks" throughout GeoServer. Each hook provides a way to plug in functionality via a script. See the *Scripting Hooks* section for details on each of the individual scripting hooks.

Scripts are located in the GeoServer data directory under a directory named `scripts`. Under this directory exist a number of other directories, one for each scripting hook:

```
GEOSERVER_DATA_DIR/
  ...
  scripts/
    apps/
    function/
    lib/
    wfs/
      tx/
    wps/
```

The `apps` directory provides an "application" hook allowing for one to provide a script invokable over http.

The `function` directory provides a Filter Function hook allowing to create new custom functions to be used for example in WFS/WMS filtering or in SLD expressions, see *Filter functions*.

The `lib` directory is not a hook but is meant to be a location where common scripts/libraries may be placed. For instance this directory may be used as a common location for data structures and utility functions that may be utilized across many different scripts.

Note: How the `lib` directory (or if it is utilized at all) is language specific.

The `wfs/tx` directory provides a WFS Transactions hook allowing to intercept WFS transaction.

The `wps` directory provides a Web Processing Service (WPS) process hook to contribute a process invokable via WPS.

See *Scripting Hooks* for more details.

Creating scripts involves either creating a script in one of these hook directories or creating it with web user interface. New scripts are picked up automatically by GeoServer without a need to ever restart the server as is the case with a pure Java GeoServer extension.

16.5.5 Scripting Hooks

This page describes all available scripting hooks. Every hook listed on this page is implemented by all the supported language extensions. However, depending on the language, the interfaces and api used to write a script may differ. Continue reading for more details.

Applications

The "app" hook provides a way to contribute scripts that are intended to be run over http. An app corresponds to a named directory under the `scripts/apps` directory. For example:

```
GEOSERVER_DATA_DIR/
  ...
  scripts/
    apps/
      hello/
```

An app directory must contain a *main* file that contains the "entry point" into the application. Every time the app is invoked via an http request this main file is executed.

The contains of the main file differ depending on the language. The default for all languages is simply that the main file contain a function named "run" that takes two arguments, the http request and response. For example, in beanshell:

```
import org.restlet.data.*;

run(request,response) {
  response.setEntity("Hello World!", MediaType.TEXT_PLAIN);
}
```

As explained above this api can differ depending on the language. For example in Python we have the well defined WSGI specification that gives us a standard interface for Python web development. The equivalent Python script to that above is:

```
def app(environ, start_response):
  start_response('200 OK', [('Content-Type', 'text/plain')])
  return ['Hello World!']
```

For the JavaScript app hook, scripts are expected to export an `app` function that conforms to the JSGI specification (v0.3). The equivalent 'Hello World' app in JavaScript would look like the following (in `/scripts/apps/hello/main.js`):

```
exports.app = function(request) {
  return {
    status: 200,
    headers: {"Content-Type": "text/plain"},
    body: ["Hello World"]
  }
};
```

Applications are http accessible at the path `/script/apps/{app}` where `{app}` is the name of the application. For example assuming a local GeoServer the url for for the application would be:

```
http://localhost:8080/geoserver/script/apps/hello
```

Warning: Because of security risks the path will not be accessible if the default admin password has not been changed.

Web Processing Service

The wps hook provides a way to provides a way to contribute scripts runnable as a WPS process. The process is invoked using the standard WPS protocol the same way an existing well-known process would be.

All processes are located under the `scripts/wps` directory. Each process is located in a file named for the process. For example:

```
GEOSERVER_DATA_DIR/
  ...
  scripts/
    wps/
      buffer.bsh
```

The process will be exposed using the extension as the namespace prefix, and the file name as the process name, for example, the above process will show up as `bsh:buffer`. It is also possible to put scripts in subdirectories of `script/wps`, in this case the directory name will be used as the process namespace, for example:

```
GEOSERVER_DATA_DIR/
  ...
  scripts/
    wps/
      foo/
        buffer.bsh
```

will expose the process as `foo:buffer`.

A process script must define two things:

1. The process metadata: title, description, inputs, and outputs
2. The process routine itself

The default for languages is to define the metadata as global variables in the script and the process routine as a function named "run". For example, in groovy:

```
import org.locationtech.jts.geom.Geometry

title = 'Buffer'
description = 'Buffers a geometry'

inputs = [
  geom: [name: 'geom', title: 'The geometry to buffer', type: Geometry.class],
  distance: [name: 'distance', title: 'The buffer distance', type: Double.class]
]

outputs = [
  result: [name: 'result', title: 'The buffered geometry',  type: Geometry.class]
]

def run(input) {
  return [result: input.geom.buffer(input.distance)]
}
```

In Python the api is slightly different and makes use of Python decorators:

```
from geoserver.wps import process
from org.locationtech.jts.geom import Geometry

@process(
  title='Buffer',
  description='Buffers a geometry',
  inputs={
    'geom': (Geometry, 'The geometry to buffer'),
    'distance':(float,'The buffer distance')
  },
  outputs={
    'result': (Geometry, 'The buffered geometry')
  }
)
def run(geom, distance):
  return geom.buffer(distance);
```

In JavaScript, a script exports a `process` object (see the GeoScript JS API docs for more detail) in order to be exposed as a WPS process. The following is an example of a simple buffer process (saved in `scripts/wps/buffer.js`):

```
var Process = require("geoscript/process").Process;

exports.process = new Process({
  title: "JavaScript Buffer Process",
  description: "Process that buffers a geometry.",
  inputs: {
    geom: {
      type: "Geometry",
      title: "Input Geometry",
      description: "The target geometry."
    },
```

```
    distance: {
      type: "Double",
      title: "Buffer Distance",
      description: "The distance by which to buffer the geometry."
    }
  },
  outputs: {
    result: {
      type: "Geometry",
      title: "Result",
      description: "The buffered geometry."
    }
  },
  run: function(inputs) {
    return {result: inputs.geom.buffer(inputs.distance)};
  }
});
```

Once implemented a process is invoked using the standard WPS protocol. For example assuming a local GeoServer the url to execute the process would be:

```
http://localhost:8080/geoserver/wps
  ?service=WPS
  &version=1.0.0
  &request=Execute
  &identifier=XX:buffer
  &datainputs=geom=POINT(0 0)@mimetype=application/wkt;distance=10
```

(Substitue `XX:buffer` for the script name followed by the extension. E.g. `py:buffer` for Python or `js:buffer` for JavaScript.)

Filter Functions

The Filter Functions hook provides a way to create new Filter Function. These functions may be used, for example, in WFS/WMS filtering or in SLD expressions, for more information about Filter Functions see *Filter functions*. GeoServer already provides many built in functions, for a complete list see *Filter Function Reference*.

All created functions are located under the `scripts/function` directory. For creating new functions use *Scripting Web User Interface* or place directly function file in `scripts/function` directory, for example, to create a function named `camelcase` using the python language create file `scripts/function/camelcase.py`.

The contains of the function file differ depending on the language. The default for all languages is simply that the function file contains a function named "run". For example, in python:

```
def run(value, args):
  return ''.join(x for x in args[0].title() if not x.isspace())
```

The filter function name equals the function file name, for example, if there is `scripts/function/camelcase.py` file then it can be used in SLD like this:

```
...
<TextSymbolizer>
       <Label>
          <ogc:Function name="camelcase">
            <ogc:PropertyName>STATE_NAME</ogc:PropertyName>
```

```
          </ogc:Function>
        </Label>
  ...
</TextSymbolizer>
...
```

WFS Transactions

WFS Transactions hook provides a way one can intercept WFS Transactions. It could be used, for example, to add validation or fill some attributes based on other ones.

All created WFS Transactions hooks are located under the `scripts/wfs/tx` directory. For creating new functions use *Scripting Web User Interface* or place file directly in `scripts/wfs/tx` directory. The file name does not matter in WFS Transaction hook.

To intercept transaction one should declare a method with name specific to transaction phase, for example, to manipulate data before update use `preUpdate`. Available methods in python are:

```
from geoserver.wfs import tx

def before(req, context):
  context['before'] = True

def preInsert(inserted, req, context):
  context['preInsert'] = True

def postInsert(inserted, req, context):
  context['postInsert'] = True

def preUpdate(updated, props, req, context):
  context['preUpdate'] = True

def postUpdate(updated, props, req, context):
  context['postUpdate'] = True

def preDelete(deleted, req, context):
  context['preDelete'] = True

def postDelete(deleted, req, context):
  context['postDelete'] = True

def preCommit(req, context):
  context['preCommit'] = True

def postCommit(req, res, context):
  context['postCommit'] = True

def abort(req, res, context):
  context['abort'] = True
```

For example, to disallow feature deleting in python, create script:

```
from org.geoserver.wfs import WFSException

def preDelete(deleted, req, context):
  raise WFSException("It is not allowed to delete Features in this layer!")
```

16.5.6 Scripting Reference

Python

GeoServer Python API Documentation

Script Hooks

app

In Python the app hook is based on WSGI which provides a common interface for Python web application development. This is not a comprehensive introduction to WSGI, that can be found here, but the app script must provide a function named `app` that takes a dictionary containing information about the environment, and a function to start the response.

```
def app(environ, start_response):
  # do stuff here
```

The function must be present in a file named `main.py` in a named *application directory*. Application directories live under the `scripts/apps` directory under the root of the data directory:

```
GEOSERVER_DATA_DIR/
  ...
  scripts/
    apps/
      app1/
        main.py
        ...
      app2/
        main.py
        ...
```

The application is web accessible from the path `/script/apps/{app}` where `{app}` is the name of the application. All requests that start with this path are dispatched to the `app` function in `main.py`.

Hello World Example

In this example a simple "Hello World" application is built. First step is to create a directory for the app named `hello`:

```
cd $GEOSERVER_DATA_DIR/scripts/apps
mkdir hello
```

Next step is to create the `main.py` file:

```
cd hello
touch main.py
```

Next the `app` function is created and stubbed out:

```
def app(environ, start_response):
  pass
```

Within the app function the following things will happen:

1. Report an HTTP status code of 200
2. Declare the content type of the response, in this case "text/plain"
3. Generate the response, in this case the string "Hello World"

Steps 1 and 2 are accomplished by invoking the `start_response` function:

```
start_response('200 OK', [('Content-Type', 'text/plain')])
```

Step 3 is achieved by returning an array of string content:

```
return ['Hello World']
```

The final completed version:

```
def app(environ, start_response):
  start_response('200 OK', [('Content-Type', 'text/plain')])
  return ['Hello World!']
```

Note: WSGI allows for additional methods of generating responses rather than returning an array. In particular it supports returning an generator for the response content. Consult the WSGI documentation for more details.

wps

In Python the wps/process interface is much like the other languages, with a few differences. A process is defined with a function named `run` that is decorated with the `geoserver.wps.process` decorator:

```
from geoserver.wps import process

@process(...)
def run(...):
   # do something
```

The function is located in a file under the `scripts/wps` directory under the root of the data directory. A WPS process requires metadata to describe itself to the outside world including:

- A **name** identifying the process
- A short **title** describing the process
- An optionally longer **description** that describes the process in more detail
- A dictionary of **inputs** describing the parameters the process accepts as input
- A dictionary of **outputs** describing the results the process generates as output

In python the `name` is implicitly derived from the name of the file that contains the process function. The rest of the metadata is passed in as arguments to the `process` decorator. The `title` and `description` are simple strings:

```
@process(title='Short Process Title',
         description='Longer and more detailed process description')
def run():
   pass
```

The `inputs` metadata is a dictionary keyed with strings matching the names of the process inputs. The values of the dictionary are tuples in which the first element is the type of the input and the second value is the description of the input. The keys of the dictionary must match those declared in the process function itself:

```
@process(
  ...
  inputs={'arg1': (<arg1 type>, 'Arg1 description'),
          'arg2': (<arg2 type>, 'Arg2 description')}
)
def run(arg1, arg2):
  pass
```

Optionally, the input tuples can also host a third argument, a dictionary hosting more input metadata. Currently the following metadata are supported:

- min: mininum number of occurrences for the input, 0 if the input is optional
- max: maximum number of occurrences for the input, if greater than one the process will receive a list
- domain: the list of values the input can receive, which will be advertised in the WPS DescribeProcess output

For example:

```
@process(
  inputs={'geom': (Geometry, 'The geometry to buffer'),
          'distance':(float, 'The buffer distance'),
          'capStyle': (str, 'The style of buffer endings',
                       {'min': 0, 'domain' :  ('round', 'flat', 'square')}),
          'quadrantSegments': (int, 'Number of segments' , {'min': 0})}
```

Finally, the default values assigned to the `run` function parameter will show up in the capabilities document as the parameter default value:

```
@process(...)
def run(a, b, c='MyDefaultValue')
```

The `outputs` metadata is the same structure as the inputs dictionary except that for it describes the output arguments of the process:

```
@process(
  ...
  outputs={'result1': (<result1 type>, 'Result1 description'),
           'result2': (<result2 type>, 'Result2 description')}
)
def run(arg1, arg2):
  pass
```

A process must generate and return results matching the `outputs` arguments. For processes that return a single value this is implicitly determined but processes that return multiple values must be explicit by returning a dictionary of the return values:

```
@process(
  ...
  outputs={'result1': (<result1 type>, 'Result1 description'),
           'result2': (<result2 type>, 'Result2 description')}
)
def run(arg1, arg2):
```

```
  # do something
  return {
    'result1': ...,
    'result2': ...
  }
```

Buffer Example

In this example a simple buffer process is created. First step is to create a file named `buffer.py` in the `scripts/wps` directory:

```
cd $GEOSERVER_DATA_DIR/scripts/wps
touch buffer.py
```

Next the `run` function is created and stubbed out. The function will take two arguments:

1. A geometry object to buffer
2. A floating point value to use as the buffer value/distance

```
def run(geom, distance):
  pass
```

In order for the function to picked up it must first be decorated with the `process` decorator:

```
from geoserver.wps import process

@process(title='Buffer', description='Buffers a geometry')
def run(geom, distance):
  pass
```

Next the process inputs and outputs must be described:

```
from geoscript.geom import Geometry

@process(
   ...,
   inputs={ 'geom': (Geometry, 'The geometry to buffer'),
            'distance': (float,'The buffer distance')},
   outputs={'result': (Geometry, 'The buffered geometry')}
)
def run(geom, distance):
  pass
```

And finally writing the buffer routine which simply just invokes the `buffer` method of the geometry argument:

```
@process(...)
def run(geom, distance):
  return geom.buffer(distance)
```

In this case since the process returns only a single argument it can be returned directly without wrapping it in a dictionary.

The final completed version:

```
from geoserver.wps import process
from geoscript.geom import Geometry

@process(
   title='Buffer',
   description='Buffers a geometry',
   inputs={'geom': (Geometry, 'The geometry to buffer'),
           'distance':(float,'The buffer distance')},
   outputs={'result': (Geometry, 'The buffered geometry')}
)
def run(geom, distance):
  return geom.buffer(distance);
```

GeoScript-PY

As mentioned *previously* GeoScript provides scripting apis for GeoTools in various languages. Naturally the GeoServer Python extension comes with GeoScript Python enabled. In the buffer example above an example of importing a GeoScript class was shown.

The GeoScript Python api is documented here.

API Reference

In much the same way as GeoScript provides a convenient scripting layer on top of GeoTools the Python scripting extension provides a `geoserver` Python module that provides convenient access to some of the GeoServer internals.

The GeoServer Python api is documented *here*.

JavaScript

The GeoServer scripting extension provides a number of scripting *hooks* that allow script authors to take advantage of extension points in GeoServer.

Hooks

The App Hook

In JavaScript the app hook is based on JSGI which provides a common interface for JavaScript web application development. The app script must export a function named `app` that accepts a `request` object and returns a `response` object.

```
export.app = function(request) {
  // handle the request and return a response
}
```

The function must be exported from a file named `main.js` in a named *application directory*. Application directories live under the `scripts/apps` directory in the root of the data directory:

```
GEOSERVER_DATA_DIR/
  ...
  scripts/
    apps/
      app1/
        main.js
        ...
      app2/
        main.js
        ...
```

The application is web accessible from the path `/script/apps/{app}` where `{app}` is the name of the application. All requests that start with this path are dispatched to the `app` function in `main.js`.

Hello World Example

In this example a simple "Hello World" application is built. The first step is to create a directory for the app named `hello`:

```
cd $GEOSERVER_DATA_DIR/scripts/apps
mkdir hello
```

Next step is to create the `main.js` file:

```
cd hello
touch main.js
```

Within the app function the following things will happen:

1. Report an HTTP status code of 200
2. Declare the content type of the response, in this case "text/plain"
3. Generate the body of response, in this case the string "Hello World"

This is accomplished with the following code:

```
export.app = function(request) {
  return {
    status: 200, // step 1
    headers: {"Content-Type": "text/plain"}, // step 2
    body: ["Hello World"] // step 3
  };
};
```

The body of the response shown above is an array. In general, this can be any object with a `forEach` method. In this way, an app can returned chunked content instead of returning the entire body content at once.

The WPS Hook

In GeoScript JS, the `geoscript/process` module provides a `Process` constructor. A process object wraps a function with a title, description, and additional metadata about the inputs and outputs. With the GeoServer scripting extension, when a script exports a process, it is exposed in GeoServer via the WPS interface.

To better understand how to construct a well described process, we'll examine the parts of the previously provided `buffer.js` script:

```
var Process = require("geoscript/process").Process;

exports.process = new Process({
  title: "JavaScript Buffer Process",
  description: "Process that buffers a geometry.",
  inputs: {
    geom: {
      type: "Geometry",
      title: "Input Geometry",
      description: "The target geometry."
    },
    distance: {
      type: "Double",
      title: "Buffer Distance",
      description: "The distance by which to buffer the geometry."
    }
  },
  outputs: {
    result: {
      type: "Geometry",
      title: "Result",
      description: "The buffered geometry."
    }
  },
  run: function(inputs) {
    return {result: inputs.geom.buffer(inputs.distance)};
  }
});
```

When this script is saved in the `$GEOSERVER_DATA_DIR/scripts/wps` directory, it will be available to WPS clients with the identifier `js:buffer`. In general, the process identifier is the name of the script prefixed by the language extension.

First, the `require` function is used to pull in the `Process` constructor from the `geoscript/process` module:

```
var Process = require("geoscript/process").Process;
```

Next, a process is constructed and assigned to the `process` property of the `exports` object. This makes it available to other JavaScript modules that may want to import this process with the `require` function in addition to exposing the process to GeoServer's WPS. The title and description provide WPS clients with human readable information about what the process does.

```
exports.process = new Process({
  title: "JavaScript Buffer Process",
  description: "Process that buffers a geometry.",
```

All the work of a process is handled by the `run` method. Before clients can execute a process, they need to know some detail about what to provide as input and what to expect as output. In general, processes accept multiple inputs and may return multiple outputs. These are described by the process' `inputs` and `outputs` properties.

```
inputs: {
  geom: {
    type: "Geometry",
```

```
    title: "Input Geometry",
    description: "The target geometry."
  },
  distance: {
    type: "Double",
    title: "Buffer Distance",
    description: "The distance by which to buffer the geometry."
  }
},
```

The buffer process expects two inputs, named `geom` and `distance`. As with the process itself, each of these inputs has a human readable title and description that will be provided to WPS clients. The `type` property is a shorthand string identifying the data type of the input. See the Process API docs for more detail on supported input and output types.

```
outputs: {
  result: {
    type: "Geometry",
    title: "Result",
    description: "The buffered geometry."
  }
},
```

The buffer process provides a single output identified as `result`. As with each of the inputs, this output is described with `type`, `title`, and `description` properties.

To see what this process metadata looks like to a WPS client, call the WPS DescribeProcess method:

```
http://localhost:8080/geoserver/wps
  ?service=WPS
  &version=1.0.0
  &request=DescribeProcess
  &identifier=js:buffer
```

Finally, the `run` method is provided.

```
  run: function(inputs) {
    return {result: inputs.geom.buffer(inputs.distance)};
  }
});
```

The `run` method takes a single `inputs` argument. This object will have named properties corresponding the the client provided inputs. In this case, the `geom` property is a `Geometry` object from the `geoscript/geom` module. This geometry has a `buffer` method that is called with the provided distance. See the Geometry API docs for more detail on available geometry properties and methods.

The `run` method returns an object with properties corresponding to the above described outputs - in this case, just a single `result` property.

To see the results of this processs in action, call the WPS Execute method:

```
http://localhost:8080/geoserver/wps
  ?service=WPS
  &version=1.0.0
  &request=Execute
  &identifier=js:buffer
  &datainputs=geom=POINT(0 0)@mimetype=application/wkt;distance=10
```

GeoScript JS

To provide a JavaScript interface for data access and manipulation via GeoTools, the GeoServer scripting extension includes the GeoScript JS library. To best leverage the scripting hooks in GeoServer, read through the GeoScript JS API docs for detail on scripting access to GeoTools functionality with JavaScript.

GeoServer JavaScript Reference

In much the same way as GeoScript JS provides a convenient set of modules for scripting access to GeoTools, the GeoServer scripting extension includes a `geoserver` JavaScript module that allows convenient access to some of the GeoServer internals. See the *GeoServer JavaScript API Documentation* for more detail.

GeoServer JavaScript API Documentation

The scripting extension includes a `geoserver/catalog` module that allows scripts to access resources in the GeoServer catalog.

The `catalog` module

```
var catalog = require("geoserver/catalog");
```

Properties

`namespaces`
`Array` A list of namespace objects. Namespaces have `alias` and `uri` properties.

```
catalog.namespaces.forEach(function(namespace) {
  // do something with namespace.alias or namespace.uri
});
```

Methods

`getVectorLayer`(*id*)

Parameters **`id`** – `String` The fully qualified feature type identifier (e.g. "topp:states")

Returns `geoscript.layer.Layer`

Access a feature type in the catalog as a GeoScript Layer.

```
var states = catalog.getVectorLayer("topp:states");
```

16.5.7 Scripting Rest API

Like other modules in GeoServer, you can add, update, read, and delete scripts using a restful interface.

Warning: The scripting rest API will not work until you have changed the GeoServer default admin password.

WPS Scripts

`/scripts/wps[.<format>]`

Method	Action	Status code	Formats	Default Format
GET	List all scripts	200	HTML, XML, JSON	HTML

List WPS Scripts

curl -u username:password -XGET -H "Accept: text/xml" http://localhost:8080/geoserver/rest/scripts/wps

`/scripts/wps/<script.ext>`

Method	Action	Status code	Formats	Default Format
GET	Get the contents of a script	200	Text	Text
PUT	Add a new script	200	Text	Text
PUT	Update an existing script	200	Text	Text
DELETE	Delete an existing script	200		

Get a WPS Script

curl -u username:password -XGET -H "Accept: text/xml" http://localhost:8080/geoserver/rest/scripts/wps/buffer.groov

Add a WPS Script

curl -u username:password -XPUT -H "Content-type: text/plain" –data-binary @buffer.groovy http://localhost:8080/geoserver/rest/scripts/wps/buffer.groovy

Update a WPS Script

curl -u username:password -XPUT -H "Content-type: text/plain" –data-binary @buffer.groovy http://localhost:8080/geoserver/rest/scripts/wps/buffer.groovy

Delete a WPS Script

curl -u username:password -XDELETE http://localhost:8080/geoserver/rest/scripts/wps/buffer.groovy

Filter Function Scripts

/scripts/function[.<format>]

Method	Action	Status code	Formats	Default Format
GET	List all scripts	200	HTML, XML, JSON	HTML

List Function Scripts

curl -u username:password -XGET -H "Accept: text/xml" http://localhost:8080/geoserver/rest/scripts/function

/scripts/function/<script.ext>

Method	Action	Status code	Formats	Default Format
GET	Get the contents of a script	200	Text	Text
PUT	Add a new script	200	Text	Text
PUT	Update an existing script	200	Text	Text
DELETE	Delete an existing script	200		

Get a Function Script

curl -u username:password -XGET -H "Accept: text/xml" http://localhost:8080/geoserver/rest/scripts/function/buffered

Add a Function Script

curl -u username:password -XPUT -H "Content-type: text/plain" –data-binary @bufferedCentroid.groovy
http://localhost:8080/geoserver/rest/scripts/function/bufferedCentroid.groovy

Update a Function Script

curl -u username:password -XPUT -H "Content-type: text/plain" –data-binary @bufferedCentroid.groovy
http://localhost:8080/geoserver/rest/scripts/function/bufferedCentroid.groovy

Delete a Function Script

curl -u username:password -XDELETE http://localhost:8080/geoserver/rest/scripts/function/bufferedCentroid.groovy

WFS Transaction Scripts

/scripts/wfs/tx[.<format>]

Method	Action	Status code	Formats	Default Format
GET	List all scripts	200	HTML, XML, JSON	HTML

List WFSTX Scripts

curl -u username:password -XGET -H "Accept: text/json" http://localhost:8080/geoserver/rest/scripts/wfs/tx

/scripts/wfs/tx/<script.ext>

Method	Action	Status code	Formats	Default Format
GET	Get the contents of a script	200	Text	Text
PUT	Add a new script	200	Text	Text
PUT	Update an existing script	200	Text	Text
DELETE	Delete an existing script	200		

Get a WFSTX Script

curl -u username:password -XGET -H "Accept: text/xml" http://localhost:8080/geoserver/rest/scripts/wfs/tx/check.groo

Add a WFSTX Script

curl -u username:password -XPUT -H "Content-type: text/plain" –data-binary @check.groovy http://localhost:8080/geoserver/rest/scripts/wfs/tx/check.groovy

Update a WFSTX Script

curl -u username:password -XPUT -H "Content-type: text/plain" –data-binary @check.groovy http://localhost:8080/geoserver/rest/scripts/wfs/tx/check.groovy

Delete a WFSTX Script

curl -u username:password -XDELETE http://localhost:8080/geoserver/rest/scripts/wfs/tx/check.groovy

Application Scripts

/scripts/apps/[.<format>]

Method	Action	Status code	Formats	Default Format
GET	List all scripts	200	HTML, XML, JSON	HTML

List App

curl -u username:password -XGET -H "Accept: text/xml" http://localhost:8080/geoserver/rest/scripts/apps

`/scripts/apps/<name>/main.<ext>`

Method	Action	Status code	Formats	Default Format
GET	Get the contents of a script	200	Text	Text
PUT	Add a new script	200	Text	Text
PUT	Update an existing script	200	Text	Text
DELETE	Delete an existing script	200		

Get an App

curl -u username:password -XGET -H "Accept: text/xml" http://localhost:8080/geoserver/rest/scripts/apps/buffer/mair

Add a App Script

curl -u username:password -XPUT -H "Content-type: text/plain" –data-binary @app_buffer.groovy http://localhost:8080/geoserver/rest/scripts/apps/buffer/main.groovy

Update a App Script

curl -u username:password -XPUT -H "Content-type: text/plain" –data-binary @app_buffer.groovy http://localhost:8080/geoserver/rest/scripts/apps/buffer/main.groovy

Delete a Add Script

curl -u username:password -XDELETE http://localhost:8080/geoserver/rest/scripts/apps/buffer/main.groovy

Scripting Sessions

`/scripts/sessions[.<format>]`

Method	Action	Status code	Formats	Default Format
GET	List all scripts	200	JSON	JSON

List Scripting Sessions

curl -u username:password -XGET -H "Accept: text/json" http://localhost:8080/geoserver/rest/sessions

`/scripts/sessions/<language>/<id>`

Method	Action	Status code	Formats	Default Format
GET	Get the scripting session	200	JSON	JSON
POST	Create a scripting session	200	TEXT	TExT
PUT	Run a script	200	Text	Text

Get a Scripting Session

curl -u username:password -XGET -H "Accept: text/json" http://localhost:8080/geoserver/rest/sessions/groovy/0

Create a Scripting Session

curl -u username:password -XPOST http://localhost:8080/geoserver/rest/sessions/groovy

Run a Script in a Session

curl -u username:password -XPUT –data-binary @script.groovy http://localhost:8080/geoserver/rest/sessions/groovy/0

16.6 SpatiaLite

SpatiaLite is the spatial extension of the popular SQLite embedded relational database.

Note: GeoServer does not come built-in with support for SpatiaLite; it must be installed through an extension. Furthermore it requires that additional native libraries be available on the system. Proceed to *Installing the SpatiaLite extension* for installation details.

16.6.1 SpatiaLite version

The GeoServer SpatiaLite extension includes its own versions of SQLite (3.7.2) and SpatiaLite (2.4.0) and therefore these libraries need not be installed on the system in order to use the extension. However this internal version of SpatiaLite is compiled against the PROJ and GEOS libraries so they must be installed on the system in order for the extension to function. See *Native Libraries* for more details.

16.6.2 Supported platforms

This extension is supported for Windows, Linux, and Mac OS X. Both 32-bit and 64-bit platforms are supported. For Mac OS X only Intel based machines are supported (ie. not PowerPC).

16.6.3 Installing the SpatiaLite extension

1. Download the SpatiaLite extension from the nightly GeoServer community module builds.

   Warning: Make sure to match the version of the extension to the version of the GeoServer instance.

2. Extract the contents of the archive into the `WEB-INF/lib` directory of the GeoServer installation.
3. Ensure the native library dependencies are satisfied.

16.6.4 Native Libraries

The version of SpatiaLite included in the extension is compiled against the GEOS and PROJ libraries so they must be installed on the system. If the libraries are not installed on the system the extension will not functionand remain disabled.

Note: Pre-compiled libraries are available for the following platforms and can be found here.

In general if the libraries are installed in a "default" location then they should be picked up by java with no problem. However some systems will require further configuration that differs based on operating system.

Windows

The DLL's must be copied into the `C:\WINDOWS\system32` directory.

Linux

If the libraries are not installed in a default search location like `/usr/lib` then the `LD_LIBRARY_PATH` environment variable must be set and visible to Java.

Mac OS X

Same as Linux except that the `DYLD_LIBRARY_PATH` environment variable is used.

16.6.5 Adding a SpatiaLite database

Once the extension is properly installed `SpatiaLite` will show up as an option when creating a new data store.

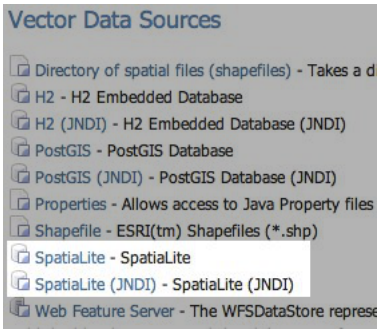


Fig. 16.1: *SpatiaLite in the list of vector data sources*

Fig. 16.2: *Configuring a SpatiaLite data store*

16.6.6 Configuring a SpatiaLite data store

`database`	The name of the database to connect to. See *notes* below.
`schema`	The database schema to access tables from. Optional.
`user`	The name of the user to connect to the database as. Optional.
`password`	The password to use when connecting to the database. Optional, leave blank for no password.
`max connections` `min connections`	Connection pool configuration parameters. See the *Database Connection Pooling* section for details.

The *database* parameter may be specified as an absolute path or a relative one. When specified as a relative path the database will created in the `spatialite` directory, located directly under the root of the GeoServer data directory.

16.7 Dynamic colormap generation

`ras:DynamicColorMap` is a **Raster-to-Raster** rendering transformation which applies a dynamic color map to a Raster on top of its statistics and a set of colors.

16.7.1 Installing the dynamic colormap community extension

1. Download the extension from the nightly GeoServer community module builds.

Warning: Make sure to match the version of the extension to the version of the GeoServer instance!

2. Extract the contents of the archive into the `WEB-INF/lib` directory of the GeoServer installation.

16.7.2 Usage

The following SLD invokes a Dynamic Color Map rendering transformation on a Coverage using colorMaps created on top of QuantumGIS SVG files. Dynamic Color Map Rendering Transformation takes data as first parameter (the coverage) and ColorRamp as second parameter which is a colorMap.

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <StyledLayerDescriptor version="1.0.0"
3      xsi:schemaLocation="http://www.opengis.net/sld StyledLayerDescriptor.xsd"
4      xmlns="http://www.opengis.net/sld"
5      xmlns:ogc="http://www.opengis.net/ogc"
6      xmlns:xlink="http://www.w3.org/1999/xlink"
7      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
8    <NamedLayer>
9      <Name>DynamicColorMap</Name>
10     <UserStyle>
11       <Title>DynamicColorMap</Title>
12       <Abstract>A DynamicColorMap</Abstract>
13       <FeatureTypeStyle>
14         <Transformation>
15           <ogc:Function name="ras:DynamicColorMap">
16             <ogc:Function name="parameter">
17               <ogc:Literal>data</ogc:Literal>
18             </ogc:Function>
19             <ogc:Function name="parameter">
20               <ogc:Literal>colorRamp</ogc:Literal>
21               <ogc:Function name="colormap">
22                 <ogc:Literal>gmt\GMT_panoply</ogc:Literal>
23                 <ogc:Function name="gridCoverageStats"><ogc:Literal>minimum</
   ↪ogc:Literal></ogc:Function>
24                 <ogc:Function name="gridCoverageStats"><ogc:Literal>maximum</
   ↪ogc:Literal></ogc:Function>
25               </ogc:Function>
26             </ogc:Function>
27           </ogc:Function>
28         </Transformation>
29         <Rule>
30          <Name>rule1</Name>
31          <RasterSymbolizer>
32            <Opacity>1.0</Opacity>
33          </RasterSymbolizer>
34         </Rule>
35       </FeatureTypeStyle>
36     </UserStyle>
37   </NamedLayer>
38  </StyledLayerDescriptor>
```

Key aspects of the SLD are:

- **Lines 14-15** define the rendering transformation, using the process `ras:DynamicColorMap`.

- **Lines 16-18** supply the input data parameter, named `data` in this process.
- **Lines 19-21** supply a value for the process's `colorRamp` parameter which specifies a colorMap.
- **Lines 22-23** supply the value for the `colorMap` parameter. In this case it's a reference to a SVG containing a LinearGradient definition.

  A sample of QuantumGIS SVG LinearGradient subelement is:

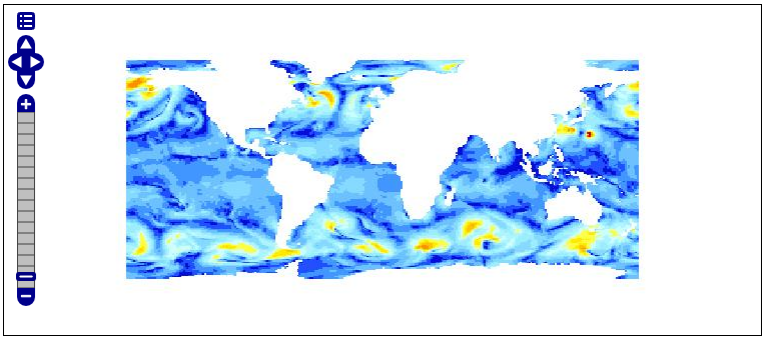
```
<linearGradient id="GMT_panoply" gradientUnits="objectBoundingBox" spreadMethod=
↪"pad" x1="0%" x2="100%" y1="0%" y2="0%">
    <stop offset="0.00%" stop-color="rgb(4,14,216)" stop-opacity="1.0000"/>
    <stop offset="6.25%" stop-color="rgb(4,14,216)" stop-opacity="1.0000"/>
    <stop offset="6.25%" stop-color="rgb(32,80,255)" stop-opacity="1.0000"/>
    <stop offset="12.50%" stop-color="rgb(32,80,255)" stop-opacity="1.0000"/>
    <stop offset="12.50%" stop-color="rgb(65,150,255)" stop-opacity="1.0000"/>
    <stop offset="18.75%" stop-color="rgb(65,150,255)" stop-opacity="1.0000"/>
    <stop offset="18.75%" stop-color="rgb(109,193,255)" stop-opacity="1.0000"/>
    <stop offset="25.00%" stop-color="rgb(109,193,255)" stop-opacity="1.0000"/>
    <stop offset="25.00%" stop-color="rgb(134,217,255)" stop-opacity="1.0000"/>
    <stop offset="31.25%" stop-color="rgb(134,217,255)" stop-opacity="1.0000"/>
    <stop offset="31.25%" stop-color="rgb(156,238,255)" stop-opacity="1.0000"/>
    <stop offset="37.50%" stop-color="rgb(156,238,255)" stop-opacity="1.0000"/>
    <stop offset="37.50%" stop-color="rgb(175,245,255)" stop-opacity="1.0000"/>
    <stop offset="43.75%" stop-color="rgb(175,245,255)" stop-opacity="1.0000"/>
    <stop offset="43.75%" stop-color="rgb(206,255,255)" stop-opacity="1.0000"/>
    <stop offset="50.00%" stop-color="rgb(206,255,255)" stop-opacity="1.0000"/>
    <stop offset="50.00%" stop-color="rgb(255,254,71)" stop-opacity="1.0000"/>
    <stop offset="56.25%" stop-color="rgb(255,254,71)" stop-opacity="1.0000"/>
    <stop offset="56.25%" stop-color="rgb(255,235,0)" stop-opacity="1.0000"/>
    <stop offset="62.50%" stop-color="rgb(255,235,0)" stop-opacity="1.0000"/>
    <stop offset="62.50%" stop-color="rgb(255,196,0)" stop-opacity="1.0000"/>
    <stop offset="68.75%" stop-color="rgb(255,196,0)" stop-opacity="1.0000"/>
    <stop offset="68.75%" stop-color="rgb(255,144,0)" stop-opacity="1.0000"/>
    <stop offset="75.00%" stop-color="rgb(255,144,0)" stop-opacity="1.0000"/>
    <stop offset="75.00%" stop-color="rgb(255,72,0)" stop-opacity="1.0000"/>
    <stop offset="81.25%" stop-color="rgb(255,72,0)" stop-opacity="1.0000"/>
    <stop offset="81.25%" stop-color="rgb(255,0,0)" stop-opacity="1.0000"/>
    <stop offset="87.50%" stop-color="rgb(255,0,0)" stop-opacity="1.0000"/>
    <stop offset="87.50%" stop-color="rgb(213,0,0)" stop-opacity="1.0000"/>
    <stop offset="93.75%" stop-color="rgb(213,0,0)" stop-opacity="1.0000"/>
    <stop offset="93.75%" stop-color="rgb(158,0,0)" stop-opacity="1.0000"/>
    <stop offset="100.00%" stop-color="rgb(158,0,0)" stop-opacity="1.0000"/>
</linearGradient>
```

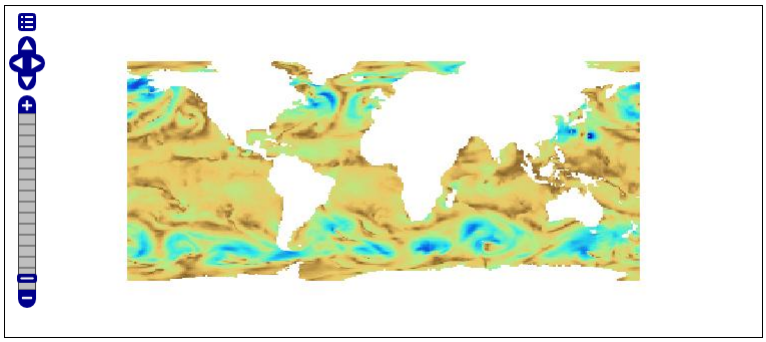
  Which should be rendered like this:

- **Lines 24** supplies the `minimum` parameter which is determined through a FilterFunction which takes the minimum value from the GridCoverage statistics,
- **Lines 25** supplies the `maximum` parameter which is determined through a FilterFunction which takes the maximum value from the GridCoverage statistics,

  The resulting image may look like this (you may note the STEPs across colors due to color intervals):

  Using an GMT_drywet SVG, the resulting image may look like this, which uses a smoother color ramp:





Alternatively, a ColorMap may be specified this way:

```
..........
           <ogc:Function name="ras:DynamicColorMap">
             <ogc:Function name="parameter">
               <ogc:Literal>data</ogc:Literal>
             </ogc:Function>
             <ogc:Function name="parameter">
               <ogc:Literal>colorRamp</ogc:Literal>
               <ogc:Function name="colormap">
                 <ogc:Literal>#0000FF;#00FF00;#FF0000</ogc:Literal>
                 <ogc:Function name="gridCoverageStats"><ogc:Literal>minimum</
↪ogc:Literal></ogc:Function>
                 <ogc:Function name="gridCoverageStats"><ogc:Literal>maximum</
↪ogc:Literal></ogc:Function>
               </ogc:Function>
             </ogc:Function>
           </ogc:Function>
...........
```

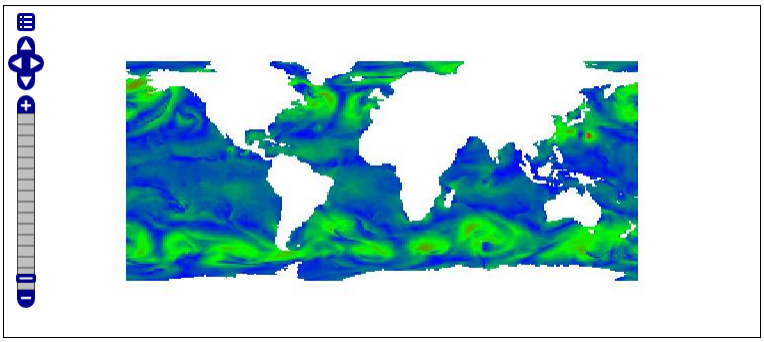
or

```
..........
           <ogc:Function name="ras:DynamicColorMap">
             <ogc:Function name="parameter">
               <ogc:Literal>data</ogc:Literal>
             </ogc:Function>
             <ogc:Function name="parameter">
               <ogc:Literal>colorRamp</ogc:Literal>
               <ogc:Function name="colormap">
                 <ogc:Literal>rgb(0,0,255);rgb(0,255,0);rgb(255,0,0)</ogc:Literal>
                 <ogc:Function name="gridCoverageStats"><ogc:Literal>minimum</
↪ogc:Literal></ogc:Function>
```

```
                <ogc:Function name="gridCoverageStats"><ogc:Literal>maximum</
↪ogc:Literal></ogc:Function>
              </ogc:Function>
            </ogc:Function>
          </ogc:Function>
...........
```

In these cases a RAMP will be used with the indicated colors.

The resulting image may look like this:



16.7.3 DynamicColorMap Requirements

- A preliminar *gdalinfo -stats* command needs to be run against the coverages in order to create the PAM Auxiliary file containing statistics and metadata.
- In order to setup colorMap from QuantumGIS, you should have copied the QuantumGIS SVG resources folder from **apps/qgis/resources/cpt-city-XXXXX** within the **GEOSERVER_DATA_DIR** as a **styles/ramps** subfolder.
- The underlying reader should support statistics retrieval by adding a PAMDataset object as a property of the returned coverage. For this reason the user should take care of setting the **CheckAuxiliaryMetadata** flag to *true* inside the *indexer.properties* or update the *.properties* file generated by GeoServer with that flag in case of already configured stores (You also need to reload the configuration in that case).

16.8 JDBCConfig

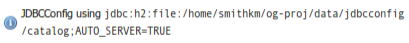
The `JDBCConfig module` enhances the scalibility performance of the GeoServer Catalog. It allows externalising the storage of the Catalog configuration objects (such as workspaces, stores, layers) to a Relational Database Management System, rather than using xml files in the *GeoServer data directory*. This way the Catalog can support access to unlimited numbers of those configuration objects efficiently.

16.8.1 Installing JDBCConfig

To install the JDBCConfig module:

1. Download the module. The file name is called `geoserver-*-jdbcconfig-plugin.zip`, where `*` is the version/snapshot name.
2. Extract this file and place the JARs in `WEB-INF/lib`.

3. Perform any configuration required by your servlet container, and then restart. On startup, JDBCConfig will create a configuration directory `jdbcconfig` in the *GeoServer data directory*.
4. Verify that the configuration directory was created to be sure installation worked then turn off GeoServer.
5. Configure JDBCConfig (*JDBCConfig configuration*), being sure to set `enabled`, `initdb`, and `import` to `true`, and to provide the connection information for an empty database.
6. Start GeoServer again. This time JDBCConfig will connect to the specified database, initialize it, import the old catalog into it, and take over from the old catalog. Subsequent start ups will skip the initialize and import steps unless you re-enable them in `jdbcconfig.properties`.
7. Log in as admin and a message should appear on the welcome page:



16.8.2 JDBCConfig configuration

The JDBCConfig module is configured in the file `jdbcconfig/jdbcconfig.properties` inside the *GeoServer data directory*. The following properties may be set:

- `enabled`: Use JDBCConfig. Turn off to use the data directory for all configuration instead.
- `initdb`: Initialize an empty database if this is set on true.
- `import` : The import configuration option tells GeoServer whether to import the current catalog from the file system to the database or not. If set to true, it will be imported and the config option will be set the value 'false' for the next start up to avoid trying to re-import the catalog configuration.
- `initScript`: Path to initialisation script .sql file. Only used if initdb = true.

JNDI

Get the database connection from the application server via JNDI lookup.

- `jndiName`: The JNDI name for the data source. Only set this if you want to use JNDI, the JDBC configuration properties may still be set for in case the JNDI Lookup fails.

Direct JDBC Connection

Provide the connection parameters directly in the configuration file. This includes the password in the clear which is a potential security risk. To avoid this use JNDI instead.

- `jdbcUrl`: JDBC direct connection parameters.
- `username`: JDBC connection username.
- `password`: JDBC connection password.
- `pool.minIdle`: minimum connections in pool
- `pool.maxActive`: maximum connections in pool
- `pool.poolPreparedStatements`: whether to pool prepared statements
- `pool.maxOpenPreparedStatements`: size of prepared statement cache, only used if pool.poolPreparedStatements = true
- `pool.testOnBorrow`: whether to validate connections when obtaining from the pool

- `pool.validationQuery`: validation query for connections from pool, must be set when pool.testOnBorrow = true

16.9 MBTiles Extension

This plugin brings in the ability to read and write MBTiles files in GeoServer. MBTiles is an SQLite based standard format that is able to hold a single tiles map layer in a file.

MBTiles can both be used as a raster input datastore as well as an WMS *GetMap* output format.

16.9.1 Installing the GeoServer MBTiles extension

> **Warning:** Make sure to install corresponding WPS extension for GeoServer instance before installing GeoServer MBTiles!

1. Download the extension from the nightly GeoServer community module builds.

> **Warning:** Make sure to match the version of the extension to the version of the GeoServer instance!

2. Extract the contents of the archive into the `WEB-INF/lib` directory of the GeoServer installation.

16.9.2 MBTiles Data Store

Adding an MBTiles Mosaic Data Store

When the extension has been installed, :guilabel:MBTiles‘ will be an option in the *Raster Data Sources* list when creating a new data store.

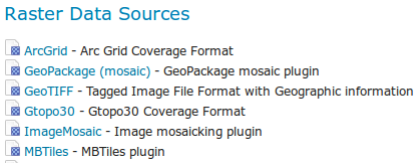


Fig. 16.3: *MBTiles in the list of raster data stores*

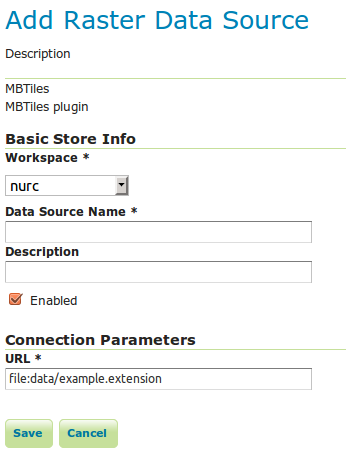


Fig. 16.4: *Configuring an MBTiles data store*

Option	Description
`Workspace`	Name of the workspace to contain the MBTiles Mosaic store. This will also be the prefix of the raster layers created from the store.
`Data Source Name`	Name of the MBTiles Store as it will be known to GeoServer. This can be different from the filename.
`Description`	A full free-form description of the MBTiles store.
`Enabled`	If checked, it enables the store. If unchecked (disabled), no data in the GeoPackage Mosaic Store will be served from GeoServer.
`URL`	Location of the MBTiles file. This can be an absolute path (such as `file:C:\Data\landbase.mbtiles`) or a path relative to GeoServer's data directory (such as `file:data/landbase.mbtiles`).

16.9.3 MBTiles Output Format

MBTiles WMS Output Format

Any WMS *GetMap* request can be returned in the form of a Geopackage by specifying `format=mbtiles` as output format (see *WMS output formats*). The returned result will be an MBTiles file with a single tile layer.

The following additional parameters can be passed on using *format_options*:

- `tileset_name`: name to be used for tileset in mbtiles file (default is name of layer(s)).
- `min_zoom`, `max_zoom`, `min_column`, `max_column`, `min_row`, `max_row`: set the minimum and maximum zoom level, column, and rows
- `gridset`: name of gridset to use (otherwise default for CRS is used)

MBTiles WPS Process

It is possible to generate an `mbtiles` file by calling the WPS process `gs:MBTiles`. This process requires the following parameters:

- `layername`: Name of the input layer.

- `format` : format of the final images composing the file.
- `minZoom`, `maxZoom`, `minColumn`, `maxColumn`, `minRow`, `maxRow`: *(Optional)* set the minimum and maximum zoom level, column, and rows.
- `boundingbox`: *(Optional)* Bounding box of the final mbtiles. If CRS is not set, the layer native one is used.
- `path`: *(Optional)* path of the directory where the mbtiles file is stored.
- `filename`: *(Optional)* name of the mbtiles file created.
- `bgColor`: *(Optional)* value associated to the background colour.
- `transparency`: *(Optional)* parameter indicating if the transparency must be present.
- `stylename`, `stylepath`, `stylebody`: *(Optional)* style to associate to the layer. Only one of these 3 parameters can be used.

The process returns an URL containing the path of the generated file.

16.10 GeoPackage Extension

This plugin brings in the ability to write GeoPackage files in GeoServer. Reading GeoPackage files is part of the core functionality of GeoServer, and does not require this extension.

GeoPackage is an SQLite based standard format that is able to hold multiple vector and raster data layers in a single file.

GeoPackage can be used as an output format for WFS *GetFeature* (creating one vector data layer) as well as WMS *GetMap* (creating one raster data layer). The GeoServer GeoPackage extension also allows to create a completely custom made GeoPackage with multiple layers, using the GeoPackage process.

16.10.1 Installing the GeoServer GeoPackage extension

1. If you haven't done already, install the WPS extension: *Installing the WPS extension*.
2. Download the Geopkg extension from the nightly GeoServer community module builds.

   Warning: Make sure to match the version of the extension to the version of the GeoServer instance!

3. Extract the contents of the archive into the `WEB-INF/lib` directory of the GeoServer installation.

16.10.2 GeoPackage As Output

GeoPackage WMS Output Format

Any WMS *GetMap* request can be returned in the form of a Geopackage by specifying `format=geopackage` as output format (see *WMS output formats*). The returned result will be a GeoPackage file with a single tile layer.

The following additional parameters can be passed on using *format_options*:

- `tileset_name`: name to be used for tileset in geopackage file (default is name of layer(s)).

- `min_zoom`, `max_zoom`, `min_column`, `max_column`, `min_row`, `max_row`: set the minimum and maximum zoom level, column, and rows
- `gridset`: name of gridset to use (otherwise default for CRS is used)

GeoPackage WFS Output Format

Any WFS *GetFeature* request can be returned as a Geopackage by specifying `format=geopackage` as output format (see *WFS output formats*). The returned result will be a GeoPackage file with a single features layer.

GeoPackage WPS Process

A custom GeoPackage can be created with any number of tiles and features layers using the `GeoPackage` WPS Process (see *Process Cookbook*).

The WPS process takes in one parameter: `contents` which is an xml schema that represents the desired output.

General outline of a `contents` scheme:

```
<geopackage name="mygeopackage" xmlns="http://www.opengis.net/gpkg">

<features name="myfeaturelayer" identifier="L01">
        <description>describe the layer</description>
        <srs> EPSG:4216 </srs>
        <bbox>
                <minx>-180</minx>
                <miny>-90</miny>
                <maxx>180</maxx>
                <maxy>90</maxy>
        </bbox>
    ...
</features>

<tiles name="mytileslayer" identifier="L02">
        <description>describe the layer</description>
        <srs>..</srs>
        <bbox>..</bbox>
        ...
</tiles>

</geopackage>
```

Each geopackage has a mandatory `name`, which will be the name of the file (with the extension .gpkg added). Each layer (features or tiles) has the following properties:

- `name` (mandatory): the name of the layer in the geopackage;
- `identifier` (optional): an identifier for the layer;
- `description` (optional): a description for the layer;
- `srs` (mandatory for tiles, optional for features): coordinate reference system; for features the default is the SRS of the feature type;
- `bbox` (mandatory for tiles, optional for features): the bounding box; for features the default is the bounding box of the feature type.

Outline of the features layer:

```
<features name="myfeaturelayer" identifier="L01">
        <description>..</description>
        <srs>..</srs>
        <bbox>..</bbox>
        <featuretype>myfeaturetype</featuretype>
        <propertynames>property1, property2</propertynames>
        <filter>..</filter>
</features>
```

Each features layer has the following properties:

- `featuretype` (mandatory): the feature type
- `propertynames` (optional): list of comma-separated names of properties in feature type to be included (default is all properties)
- `filter` (optional): any OGC filter that will be applied on features before output

Outline of the tiles layer:

```
<tiles name="mytileslayer" identifier="L02">
        <description>...</description>
        <srs>..</srs>
        <bbox>..</bbox>
        <layers>layer1, layer2</styles>
        <styles> style1, style2 </styles>
        <sld> path/to/file.sld </sld>
        <sldBody> .. </sldBody>
        <format>mime/type</format>
        <bgcolor>ffffff</bgcolor>
        <transparent>true</transparent>
        <coverage>
                <minZoom>5</minZoom>
                <maxZoom>50</maxZoom>
                <minColumn>6</minColumn>
                <maxColumn>60</maxColumn>
                <minRow>7</minRow>
                <maxRow>70</maxRow>
        <coverage>
        <gridset>
                ...
        </gridset>
</tiles>
```

Each tiles layer has the following properties:

- `layers` (mandatory): comma-separated list of layers that will be included
- **`styles`, `sld`, and `sldbody` are mutually exclusive, having one is mandatory**
  - `styles`: list of comma-separated styles to be used
  - `sld`: path to sld style file
  - `sldbody`: inline sld style file
- `format` (optional): mime-type of image format of tiles (image/png or image/jpeg)
- `bgcolor` (optional): background colour as a six-digit hexadecimal RGB value
- `transparent` (optional): transparency (true or false)

- `coverage` (optional)
- `minzoom`, `maxzoom`, `minColumn`, `maxColumn`, `minRow`, `maxRow` (all optional): set the minimum and maximum zoom level, column, and rows
- `gridset` (optional): see following

Gridset can take on two possible (mutually exclusive) forms:

```
<gridset>
        <name>mygridset</name>
</gridset>
```

where the `name` of a known gridset is specified; or a custom gridset may be defined as follows:

```
<gridset>
        <grids>
                <grid>
                <zoomlevel>1</zoomlevel>
                <tileWidth>256</tileWidth>
                <tileHeight>256</tileHeight>
                <matrixWidth>4</matrixWidth>
                <matrixHeight>4</matrixHeight>
                <pixelXSize>0.17</pixelXSize>
                <pixelYSize>0.17</pizelYSize>
                </grid>
                <grid>...</grid>
                ...
</grids>
</gridset>
```

16.11 PGRaster

The PGRaster geoserver module adds the ability to simplify the configuration of a PostGis Raster based ImageMosaic-JDBC store. Before proceeding, make sure to take a look to the PostGis Raster plugin documentation for background information. Note that configuration files, table creations and raster imports explained in that documentation, will be automatically handled by this module.

This module allows to do the following steps automatically:

1. use raster2pgsql (optionally) to import raster tiles previously configured with gdal_retile
2. create a metadata table (optionally) referring to tiles tables created through raster2pgsql
3. create the imageMosaic JDBC XML configuration containing PostGis database connection parameters, attributes mapping and coverage configuration.
4. configure the imageMosaic JDBC on top of the newly configured XML.

16.11.1 Requirements

- You must have a PostGIS 2.0 database where your raster tiles will be stored.
- Raster tiles should have been previously created using `gdal_retile` since this module will simply import them and configure the store. The ImageMosaic JDBC setup example documentation provides examples of how to do that.

- In case you want to perform automatic import of the raster tiles into the database, you need to have raster2pgsql and psql executables installed on your machine and configured on your `PATH`. (In case your PostGIS 2.0 installation is on the same machine where you will run GeoServer, the executables should be already available).

16.11.2 Installation

1. Download the pgraster community module for your version of GeoServer from the download page.
2. Unzip the archive into the WEB-INF/lib directory of the GeoServer installation.

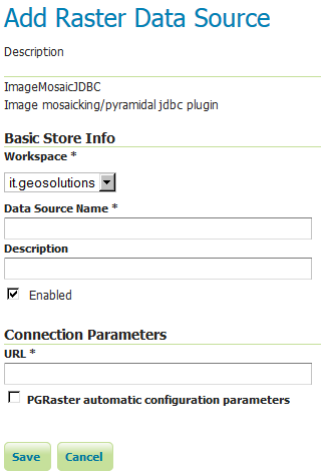
Note: On Windows, make sure to add a `RASTER2PGSQL_PATH=Drive:\Path\to\bin\folder\containing_ra` property to the `JAVA_OPTS` as an instance: `JAVA_OPTS=-DRASTER2PGSQL_PATH=C:\work\programs\PostgreS 2\bin`

---

3. Restart GeoServer.

16.11.3 Usage

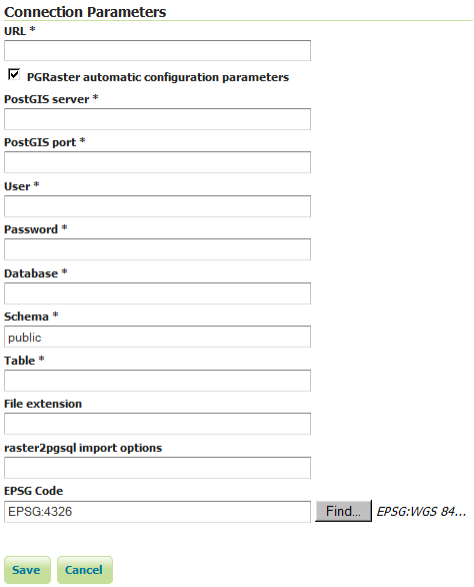
1. As for any other store configuration, go to Stores->Add new Store
2. Select ImageMosaicJDBC. You will see the usual "Add Raster Data Source" form.



For backward compatibility, you may still configure an ImageMosaicJDBC in the old-way, by specifying the URL of a valid XML configuration file, as done in the past (Where all the components of the ImageMosaicJDBC need to be configured by hand by the user).

3. Notice the presence of a checkBox which allows to proceed with the PGRaster automatic configuration parameters specification. Once Clicking on it, you will see a set of new parameters for the automatic configuration step. When enabling that checkBox, the URL parameter needs to point to the main folder containing the rasters which have been previously produced using gdal_retile.

   Other parameters are explained below:



Name	Description
PostGIS server	The PostGIS server IP
PostGIS port	The PostGIS server port
User	The PostGIS DB user
Password	The PostGIS DB password
Database	The PostGIS Database (should have already been created)
Schema	The schema where the table will be created (default is public. The schema need to be already defined into the Database before the import)
Table	The name of the metadata table which contains all the references to
File extension	The extension of the raster files to be imported (such as `png`). It may not be specified when raster tiles have been already manually imported into the database by the user
raster2pgsql import options	The raster2pgsql script importing options (as an instance "-t 128x128" for raster tiles of 128x128). It may not be specified when raster tiles have been already manually imported into the database by the user
EPSG Code	The EPSG code which will be configured in the coverage configuration xml. (Default is 4326)

16.11.4 Limitations

Right now it doesn't allow to import data folders which have been created with the gdal_retile's `useDirForEachRow` option.

16.12 WPS download community module

WPS download module provides some useful features for easily downloading: * Raster or Vector layer as zip files * Large maps as images * Time based animation The module also provides facilities to control the output file size.

16.12.1 Installing the WPS download module

1. Download the WPS download module from the nightly GeoServer community module builds.

   Warning: Make sure to match the version of the extension to the version of the GeoServer instance.

2. Extract the contents of the archive into the `WEB-INF/lib` directory of the GeoServer installation.

16.12.2 Module description

This module provides the following WPS process:

- `gs:Download` : can be used for downloading Raster and Vector Layers
- `gs:DownloadEstimator` : can be used for checking if the downloaded file does not exceeds the configured limits.
- `gs:DownloadMap`: allows to download a large map with the same composition found on the client side (eventually along with an asynchronous call)
- `gs:DownloadAnimation`: allows to download a map with the same composition found on the client side, with animation over a give set of times

16.12.3 Configuring the limits

The first step to reach for using this module is to create a new file called **download.properties** and save it in the GeoServer data directory. If the file is not present GeoServer will automatically create a new one with the default properties:

```
# Max #of features
maxFeatures=100000
#8000 px X 8000 px
rasterSizeLimits=64000000
#8000 px X 8000 px X 3 bands X 1 byte per band = 192MB
writeLimits=192000000
# 50 MB
hardOutputLimit=52428800
# STORE =0, BEST =8
compressionLevel=4
# When set to 0 or below, no limit
maxAnimationFrames=1000
```

Where the available limits are:

- `maxFeatures` : maximum number of features to download
- `rasterSizeLimits` : maximum pixel size of the Raster to read
- `writeLimits` : maximum raw raster size in bytes (a limit of how much space can a raster take in memory). For a given raster, its raw size in bytes is calculated by multiplying pixel number (raster_width x raster_height) with the accumated sum of each band's pixel sample_type size in bytes, for all bands
- `hardOutputLimit` : maximum file size to download
- `compressionLevel` : compression level for the output zip file

- `maxAnimationFrames` : maximum number of frames allowed (if no limit, the maximum execution time limits will still apply and stop the process in case there are too many)

Note: Note that limits can be changed when GeoServer is running. Periodically the server will reload the properties file.

16.12.4 The processes and their usage

The following describes the various processes, separating raw downloads from rendered downloads:

Raw data download processes

These processes allow download of vector and raster data in raw form, without rendering.

Download Estimator Process

The *Download Estimator Process* checks the size of the file to download. This process takes in input the following parameters:

- `layername` : name of the layer to check
- `ROI` : ROI object to use for cropping data
- `filter` : filter for filtering input data
- `targetCRS` : CRS of the final layer if reprojection is needed

This process will return a boolean which will be **true** if the downloaded file will not exceed the configured limits.

Download Process

The *Download Process* calls the *Download Estimator Process*, checks the file size, and, if the file does not exceed the limits, download the file as a zip. The parameters to set are

- `layerName` : the name of the layer to process/download
- `filter` : a vector filter for filtering input data(optional)
- `outputFormat` : the MIME type of the format of the final file
- `targetCRS` : the CRS of the output file (optional)
- `RoiCRS` : Region Of Interest CRS (optional)
- `ROI` : Region Of Interest object to use for cropping data (optional)
- `cropToROI` : boolean parameter to allow cropping to actual ROI, or its envelope (optional)
- `interpolation` : interpolation function to use when reprojecting / scaling raster data. Values are NEAREST (default), BILINEAR, BICUBIC2, BICUBIC (optional)
- `targetSizeX` : size X in pixels of the output (optional, applies for raster input only)
- `targetSizeY` : size Y in pixels of the output (optional, applies for raster input only)

- `selectedBands` : a set of the band indices of the original raster that will be used for producing the final result (optional, applies for raster input only)
- `writeParameters` : a set of writing parameters (optional, applies for raster input only). See *Writing parameters* below section for more details on writing parameters defintion.

The `targetCRS` and `RoiCRS` parameters are using EPSG code terminology, so, valid parameters are literals like `EPSG:4326` (if we are referring to a the Geogaphic WGS84 CRS), `EPSG:3857` (for WGS84 Web Mercator CRS), etc.

ROI Definition

A `ROI` parameter is a geometry object which can also be defined if three different forms:

- as `TEXT`, in various geometry textual formats/representations
- as `REFERENCE`, which is the textual result of an HTTP GET/POST request to a specific url
- as a `SUPPROCESS` result: the format produced as result of the process execution must be a compatible geometry textual format.

As noted above, in all above forms/cases ROI geometry is defined as text, in specific formats. These can be:

- `text/xml; subtype=gml/3.1.1`: conforming to gml specs 3.1.1
- `text/xml; subtype=gml/2.1.2`: conforming to gml specs 2.1.2
- `application/wkt`: the WKT geometry representation
- `application/json`: the JSON geometry representation
- `application/gml-3.1.1`: conforming to gml specs 3.1.1
- `application/gml-2.1.2`: conforming to gml specs 2.1.2

For example, a polygon used as ROI with the following WKT representation:

```
POLYGON (( 500116.08576537756 499994.25579707103, 500116.08576537756 500110.
1012210889, 500286.2657688021 500110.1012210889, 500286.2657688021 499994.
25579707103, 500116.08576537756 499994.25579707103 ))
```

would be represented in the following forms:

- in application/wkt: `POLYGON (( 500116.08576537756 499994.25579707103, 500116.08576537756 500110.1012210889, 500286.2657688021 500110.1012210889, 500286.2657688021 499994.25579707103, 500116.08576537756 499994.25579707103 ))`
- in application/json: `{"type":"Polygon","coordinates":[[[500116.0858, 499994.2558],[500116.0858,500110.1012],[500286.2658,500110.1012], [500286.2658,499994.2558],[500116.0858,499994.2558]]]}`
- in text/xml:`500116.08576537756,499994.25579707103 500116.08576537756, 500110.1012210889 500286.2657688021,500110.1012210889 500286.2657688021,499994.25579707103 500116.08576537756,499994.25579707103`
- in application/xml: the following xml

```
<?xml version="1.0" encoding="UTF-8"?><gml:Polygon xmlns:gml="http://www.
↪opengis.net/gml" xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xlink=
↪"http://www.w3.org/1999/xlink">
  <gml:outerBoundaryIs>
```

```
    <gml:LinearRing>
      <gml:coordinates>500116.08576537756,499994.25579707103 500116.
↪08576537756,500110.1012210889 500286.2657688021,500110.1012210889 500286.
↪2657688021,499994.25579707103 500116.08576537756,499994.25579707103</
↪gml:coordinates>
    </gml:LinearRing>
  </gml:outerBoundaryIs>
</gml:Polygon>
```

The general structure of a WPS Download request POST payload consists of two parts: the first (<wps:DataInputs>) contains the input parameters for the process, and the second (<wps:ResponseForm>) contains details about delivering the output. A typical pseudo payload is the following:

```
<?xml version="1.0" encoding="UTF-8"?><wps:Execute version="1.0.0" service=
↪"WPS" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://
↪www.opengis.net/wps/1.0.0" xmlns:wfs="http://www.opengis.net/wfs"␣
↪xmlns:wps="http://www.opengis.net/wps/1.0.0" xmlns:ows="http://www.opengis.
↪net/ows/1.1" xmlns:gml="http://www.opengis.net/gml" xmlns:ogc="http://www.
↪opengis.net/ogc" xmlns:wcs="http://www.opengis.net/wcs/1.1.1" xmlns:xlink=
↪"http://www.w3.org/1999/xlink" xsi:schemaLocation="http://www.opengis.net/
↪wps/1.0.0 http://schemas.opengis.net/wps/1.0.0/wpsAll.xsd">
 <ows:Identifier>gs:WPS_Process_Name_Here</ows:Identifier>
 <wps:DataInputs>
  <wps:Input>
   <ows:Identifier>First_Param_Name</ows:Identifier>
   <wps:Data>
     (First_Param_Data)
   </wps:Data>
  </wps:Input>
  ...
  ...
 </wps:DataInputs>
 <wps:ResponseForm>
  <wps:RawDataOutput mimeType="application/zip">
   <ows:Identifier>result</ows:Identifier>
  </wps:RawDataOutput>
 </wps:ResponseForm>
</wps:Execute>
```

Each parameter for the process is defined in its own <wps:Input> xml block. In case of simple type data, such as layerName, outputFormat, targetCRS, etc, input params xml blocks have the following form:

```
<wps:Input>
 <ows:Identifier>layerName</ows:Identifier>
 <wps:Data>
  <wps:LiteralData>nurc:Img_Sample</wps:LiteralData>
 </wps:Data>
</wps:Input>
```

Note the <wps:LiteralData> tags wrapping the parameter value. In case of geometry parameters, such as filter, ROI, the parameter's <wps:Input> block is different:

```
<wps:Input>
  <ows:Identifier>ROI</ows:Identifier>
  <wps:Data>
    <wps:ComplexData mimeType="application/wkt"><![CDATA[POLYGON (( 500116.
↪08576537756 499994.25579707103, 500116.08576537756 500110.1012210889,␣
↪500286.2657688021 500110.1012210889, 500286.2657688021 499994.25579707103,␣
↪500116.08576537756 499994.25579707103 ))]]></wps:ComplexData>
```

```
  </wps:Data>
</wps:Input>
```

Note the <wps:ComplexData> tag, the mimeType="application/wkt" parameter, and the ![CDATA[] wrapping of the actual geometry data (in textual representation), according to the selected MIME type.

Note that if the ROI parameter is defined as WKT, you will need to specify a RoiCRS input parameter as well.

In case the ROI is defined using a REFERENCE source, the input block is slightly different:

```
<wps:Input>
  <ows:Identifier>ROI</ows:Identifier>
  <wps:Reference mimeType="application/wkt" xlink:href="url_to_fetch_data"␣
↪method="GET"/>
</wps:Input>
```

Note the <wps:Reference> tag replacing <wps:ComplexData> tag, and the extra xlink:href="url_to_fetch_data" parameter, which defines the url to peform the HTTP GET request. For POST request cases, tech method is switched to POST, and a <wps:Body> tag is used to wrap POST data:

```
<wps:Reference mimeType="application/wkt" xlink:href="url_to_fetch_data"␣
↪method="POST">
  <wps:Body><![CDATA[request_body_data]]></wps:Body>
</wps:Reference>
```

Filter parameter definition

A filter parameter is a definition of a vector filter operation:

- as TEXT, in various textual formats/representations
- as REFERENCE, which is the textual result of an HTTP GET/POST request to a specific url
- as a SUBPROCESS result: the format produced as result of the process execution must be a compatible geometry textual format.

Compatible text formats for filter definitions are:

- text/xml; filter/1.0
- text/xml; filter/1.1
- text/xml; cql

For more details on filter formats/languages, one can see *Supported filter languages* and *Filter functions*. Filter parameter applies to vector data. If this is the case with input data, a sample <wps:Input> block of a filter intersecting the polygon we used earlier as an example for ROI definition would be:

```
<wps:Input>
  <ows:Identifier>filter</ows:Identifier>
  <wps:Data>
    <wps:ComplexData mimeType="text/plain; subtype=cql"><![CDATA[<Intersects>
      <PropertyName>GEOMETRY</PropertyName>
        <gml:Polygon>
          <gml:outerBoundaryIs>
            <gml:LinearRing>
```

```
                <gml:coordinates>500116.08576537756,499994.25579707103␣
↪500116.08576537756,500110.1012210889 500286.2657688021,500110.1012210889␣
↪500286.2657688021,499994.25579707103 500116.08576537756,499994.25579707103
↪</gml:coordinates>
              </gml:LinearRing>
            </gml:outerBoundaryIs>
          </gml:Polygon>
      </Intersects>]]></wps:ComplexData>
  </wps:Data>
</wps:Input>
```

Sample request

Synchronous execution

The following is a sample WPS request for processing a raster dataset. Suppose we want to use the North America sample imagery (**nurc:Img_Sample**) layer, to produce an **80x80** pixels downloadable **tiff** in **EPSG:4326**

Assuming that a local geoserver instance (setup for wps/wps-download support) is running, we issue a POST request to the url:

`http://127.0.0.1:8080/geoserver/ows?service=wps`

using the following payload:

```
<?xml version="1.0" encoding="UTF-8"?><wps:Execute version="1.0.0" service=
↪"WPS" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://
↪www.opengis.net/wps/1.0.0" xmlns:wfs="http://www.opengis.net/wfs"␣
↪xmlns:wps="http://www.opengis.net/wps/1.0.0" xmlns:ows="http://www.opengis.
↪net/ows/1.1" xmlns:gml="http://www.opengis.net/gml" xmlns:ogc="http://www.
↪opengis.net/ogc" xmlns:wcs="http://www.opengis.net/wcs/1.1.1" xmlns:xlink=
↪"http://www.w3.org/1999/xlink" xsi:schemaLocation="http://www.opengis.net/
↪wps/1.0.0 http://schemas.opengis.net/wps/1.0.0/wpsAll.xsd">
 <ows:Identifier>gs:Download</ows:Identifier>
 <wps:DataInputs>
  <wps:Input>
   <ows:Identifier>layerName</ows:Identifier>
   <wps:Data>
    <wps:LiteralData>nurc:Img_Sample</wps:LiteralData>
   </wps:Data>
  </wps:Input>
  <wps:Input>
   <ows:Identifier>outputFormat</ows:Identifier>
   <wps:Data>
    <wps:LiteralData>image/tiff</wps:LiteralData>
   </wps:Data>
  </wps:Input>
  <wps:Input>
   <ows:Identifier>targetCRS</ows:Identifier>
   <wps:Data>
    <wps:LiteralData>EPSG:4326</wps:LiteralData>
   </wps:Data>
  </wps:Input>
  <wps:Input>
   <ows:Identifier>targetSizeX</ows:Identifier>
```

```
    <wps:Data>
     <wps:LiteralData>80</wps:LiteralData>
    </wps:Data>
   </wps:Input>
   <wps:Input>
    <ows:Identifier>targetSizeY</ows:Identifier>
    <wps:Data>
     <wps:LiteralData>80</wps:LiteralData>
    </wps:Data>
   </wps:Input>
  </wps:DataInputs>
  <wps:ResponseForm>
   <wps:RawDataOutput mimeType="application/zip">
    <ows:Identifier>result</ows:Identifier>
   </wps:RawDataOutput>
  </wps:ResponseForm>
</wps:Execute>
```

More parameters (from the parameter list above) can be used, for example, we can only select bands **0 and 2** from the original raster:

```
<wps:Input>
 <ows:Identifier>bandIndices</ows:Identifier>
 <wps:Data>
  <wps:LiteralData>0</wps:LiteralData>
 </wps:Data>
</wps:Input>
<wps:Input>
 <ows:Identifier>bandIndices</ows:Identifier>
 <wps:Data>
  <wps:LiteralData>2</wps:LiteralData>
 </wps:Data>
</wps:Input>
```

Or, use a **Region Of Interest** to crop the dataset:

```
<wps:Input>
  <ows:Identifier>ROI</ows:Identifier>
  <wps:Data>
    <wps:ComplexData mimeType="application/wkt"><![CDATA["POLYGON (( 500116.
↪08576537756 499994.25579707103, 500116.08576537756 500110.1012210889,␣
↪500286.2657688021 500110.1012210889, 500286.2657688021 499994.25579707103,␣
↪500116.08576537756 499994.25579707103 ))]]></wps:ComplexData>
  </wps:Data>
</wps:Input>
<wps:Input>
  <ows:Identifier>RoiCRS</ows:Identifier>
  <wps:Data>
    <wps:LiteralData>EPSG:32615</wps:LiteralData>
  </wps:Data>
</wps:Input>
```

The result produced is a zipped file to download.

Asynchronous execution

The process can also be performed asynchronously. In this case, the second part (`wps:ResponseForm`) of the wps download payload slightly changes, by using the **storeExecuteResponse** and **status** parameters, set to **true** for the `<wps:ResponseDocument>`:

```
<wps:ResponseForm>
  <wps:ResponseDocument storeExecuteResponse="true" status="true">
    <wps:RawDataOutput mimeType="application/zip">
      <ows:Identifier>result</ows:Identifier>
    </wps:RawDataOutput>
  </wps:ResponseDocument>>
</wps:ResponseForm>
```

In case of asynchronous execution, the initial request to download data returns an xml indication that the process has successfully started:

```
<?xml version="1.0" encoding="UTF-8"?><wps:ExecuteResponse xmlns:xs="http://
↪www.w3.org/2001/XMLSchema" xmlns:ows="http://www.opengis.net/ows/1.1"␣
↪xmlns:wps="http://www.opengis.net/wps/1.0.0" xmlns:xlink="http://www.w3.
↪org/1999/xlink" xml:lang="en" service="WPS" serviceInstance="http://127.0.
↪0.1:8080/geoserver/ows?" statusLocation="http://127.0.0.1:8080/geoserver/
↪ows?service=WPS&amp;version=1.0.0&amp;request=GetExecutionStatus&amp;
↪executionId=dd0d61f5-7da3-41ed-bd3f-15311fa660ba" version="1.0.0">
  <wps:Process wps:processVersion="1.0.0">
      <ows:Identifier>gs:Download</ows:Identifier>
      <ows:Title>Enterprise Download Process</ows:Title>
      <ows:Abstract>Downloads Layer Stream and provides a ZIP.</ows:Abstract>
  </wps:Process>
  <wps:Status creationTime="2016-08-08T11:03:18.167Z">
      <wps:ProcessAccepted>Process accepted.</wps:ProcessAccepted>
  </wps:Status>
</wps:ExecuteResponse>
```

The response contains a `<wps:Status>` block indicating successfull process creation and process start time. However, the important part in this response is the **executionId=dd0d61f5-7da3-41ed-bd3f-15311fa660ba** attribute in the `<wps:ExecuteResponse>` tag. The `dd0d61f5-7da3-41ed-bd3f-15311fa660ba` ID can be used as a reference for this process, in order to issue new GET requests and to check process status. These requests have the form:

```
http://127.0.0.1:8080/geoserver/ows?service=WPS&request=GetExecutionStatus&executionId=277e
```

When issued (and process has finished on the server), this GET request returns the result to download/process as a base64 encoded zip:

```
<?xml version="1.0" encoding="UTF-8"?>
<wps:ExecuteResponse xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:ows=
↪"http://www.opengis.net/ows/1.1" xmlns:wps="http://www.opengis.net/wps/1.0.
↪0" xmlns:xlink="http://www.w3.org/1999/xlink" xml:lang="en" service="WPS"␣
↪serviceInstance="http://127.0.0.1:8080/geoserver/ows?" statusLocation=
↪"http://127.0.0.1:8080/geoserver/ows?service=WPS&amp;version=1.0.0&amp;
↪request=GetExecutionStatus&amp;executionId=0c596a4d-7ddb-4a4e-bf35-
↪4a64b47ee0d3" version="1.0.0">
  <wps:Process wps:processVersion="1.0.0">
      <ows:Identifier>gs:Download</ows:Identifier>
      <ows:Title>Enterprise Download Process</ows:Title>
      <ows:Abstract>Downloads Layer Stream and provides a ZIP.</ows:Abstract>
  </wps:Process>
```

```
  <wps:Status creationTime="2016-08-08T11:18:46.015Z">
      <wps:ProcessSucceeded>Process succeeded.</wps:ProcessSucceeded>
  </wps:Status>
  <wps:ProcessOutputs>
      <wps:Output>
          <ows:Identifier>result</ows:Identifier>
          <ows:Title>Zipped output files to download</ows:Title>
          <wps:Data>
              <wps:ComplexData encoding="base64" mimeType="application/zip">
↪UEsDBBQACAgIAFdyCEkAAAAAAAAAAAAAAAApAAAAMGEwYmJkYmQtMjdkNi00...(more␣
↪zipped raster data following, ommited for space saving)...</
↪wps:ComplexData>
          </wps:Data>
      </wps:Output>
  </wps:ProcessOutputs>
</wps:ExecuteResponse>
```

Asynchronous execution (output as a reference)

The <wps:ResponseForm> of the previous asynchronous request payload example can be modified to get back a link to the file to be downloaded instead of the base64 encoded data.

```
...
<wps:ResponseForm>
  <wps:ResponseDocument storeExecuteResponse="true" status="true">
    <wps:Output asReference="true" mimeType="application/zip">
      <ows:Identifier>result</ows:Identifier>
    </wps:Output>
  </wps:ResponseDocument>
</wps:ResponseForm>
```

Note <wps:ResponseDocument> contains a <wps:Output> instead of a <wps:RawDataOutput> being used by previous example. Moreover the attribute **asReference** set to **true** has been added to the <wps:Output>.

This time, when issued (and process has finished on the server), the GET request returns the result to download as a link as part of <wps:Output><wps:Reference> .

```
<?xml version="1.0" encoding="UTF-8"?>
  <wps:ExecuteResponse xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:ows=
↪"http://www.opengis.net/ows/1.1" xmlns:wps="http://www.opengis.net/wps/1.0.
↪0" xmlns:xlink="http://www.w3.org/1999/xlink" xml:lang="en" service="WPS"␣
↪serviceInstance="http://127.0.0.1:8080/geoserver/ows?" statusLocation=
↪"http://127.0.0.1:8080/geoserver/ows?service=WPS&amp;version=1.0.0&amp;
↪request=GetExecutionStatus&amp;executionId=c1074100-446a-4963-94ad-
↪cbbf8b8a7fd1" version="1.0.0">
  <wps:Process wps:processVersion="1.0.0">
    <ows:Identifier>gs:Download</ows:Identifier>
    <ows:Title>Enterprise Download Process</ows:Title>
    <ows:Abstract>Downloads Layer Stream and provides a ZIP.</ows:Abstract>
  </wps:Process>
  <wps:Status creationTime="2016-08-08T11:38:34.024Z">
    <wps:ProcessSucceeded>Process succeeded.</wps:ProcessSucceeded>
  </wps:Status>
  <wps:ProcessOutputs>
    <wps:Output>
```

```
      <ows:Identifier>result</ows:Identifier>
      <ows:Title>Zipped output files to download</ows:Title>
      <wps:Reference href="http://127.0.0.1:8080/geoserver/ows?service=WPS&
↪amp;version=1.0.0&amp;request=GetExecutionResult&amp;executionId=c1074100-
↪446a-4963-94ad-cbbf8b8a7fd1&amp;outputId=result.zip&amp;
↪mimetype=application%2Fzip" mimeType="application/zip" />
    </wps:Output>
  </wps:ProcessOutputs>
</wps:ExecuteResponse>
```

Writing parameters

The `writeParameters` input element of a process execution allows to specify parameters to be applied by the `outputFormat` encoder when producing the output file. Writing parameters are listed as multiple `<dwn:Parameter key="writingParameterName">value</dwn:Parameter>` within a `<dwn:Parameters>` parent element. See the below xml containing full syntax of a valid example for TIFF output format:

```
<wps:Input>
  <ows:Identifier>writeParameters</ows:Identifier>
    <wps:Data>
      <wps:ComplexData xmlns:dwn="http://geoserver.org/wps/download">
        <dwn:Parameters>
          <dwn:Parameter key="tilewidth">128</dwn:Parameter>
          <dwn:Parameter key="tileheight">128</dwn:Parameter>
          <dwn:Parameter key="compression">JPEG</dwn:Parameter>
          <dwn:Parameter key="quality">0.75</dwn:Parameter>
        </dwn:Parameters>
      </wps:ComplexData>
    </wps:Data>
</wps:Input>
```

GeoTIFF/TIFF supported writing parameters

The supported writing parameters are:

- `tilewidth` : Width of internal tiles, in pixels
- `tileheight` : Height of internal tiles, in pixels
- `compression` : Compression type used to store internal tiles. Supported values are:
  - `CCITT RLE` (Lossless) (Huffman)
  - `LZW` (Lossless)
  - `JPEG` (Lossy)
  - `ZLib` (Lossless)
  - `PackBits` (Lossless)
  - `Deflate` (Lossless)
- `quality` : Compression quality for lossy compression (JPEG). Value is in the range [0 : 1] where 0 is for worst quality/higher compression and 1 is for best quality/lower compression

- `writenodata` : Supported value is one of true/false. Note that, by default, a nodata TAG is produced as part of the output GeoTIFF file as soon as a nodata is found in the GridCoverage2D to be written. Therefore, not specifying this parameter will result into writing nodata to preserve default behavior. Setting it to false will avoid writing that TAG.

Rendered map/animation download processes

These processes allow download large maps and animations.

The rendered download processes

The map and animation downloads work off a set of common parameters:

- `bbox` : a geo-referenced bounding box, controlling both output area and desired projection
- `decoration` : the name of a decoration layout to be added on top of the map
- `time` : a WMS `time` specification used to drive the selection of times across the layers in the map, and to control the frame generation in the animation
- `width` and `height` : size of the output map/animation (and in combination with bounding box, also controls the output map scale)
- `layer`: a list of layer specifications, from a client side point of view (thus, a layer can be composed of multiple server side layers)

The layer specification

A layer specification is a XML structure made of three parts:

- Name: a comma separated list of layer names (eventually just one)
- Capabilities: link to a capabilities document (optional, used when targetting remote WMS layers)
- Parameter (key, value): an extra parameter to be added in the WMS request represented by this layer (e.g., `elevation`, `CQL_FILTER`, `env`)

For example:

```
<wps:ComplexData xmlns:dwn="http://geoserver.org/wps/download">
  <dwn:Layer>
    <dwn:Capabilities>http://demo.geo-solutions.it/geoserver/ows?service=wms&amp;
↪version=1.1.1&amp;request=GetCapabilities</dwn:Name>
    <dwn:Name>topp:states</dwn:Name>
    <dwn:Parameter key="CQL_FILTER"><![CDATA[PERSONS > 1000000]]></dwn:Parameter>
  </dwn:Layer>
</wps:ComplexData>
```

Sample DownloadMap requests

The map download process has only the basic inputs described above, the `time` parameter is optional. The map download process uses the WMS machinery to produce the output, but it's not subject to the WMS service limits (width and height in this process can be limited using the WPS process security).

A download map issued against a set of local layers can look as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<wps:Execute version="1.0.0" service="WPS"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns=
↪"http://www.opengis.net/wps/1.0.0"
             xmlns:wfs="http://www.opengis.net/wfs" xmlns:wps="http://www.
↪opengis.net/wps/1.0.0"
             xmlns:ows="http://www.opengis.net/ows/1.1" xmlns:gml="http://
↪www.opengis.net/gml"
             xmlns:ogc="http://www.opengis.net/ogc" xmlns:wcs="http://www.
↪opengis.net/wcs/1.1.1"
             xmlns:xlink="http://www.w3.org/1999/xlink"
             xsi:schemaLocation="http://www.opengis.net/wps/1.0.0 http://
↪schemas.opengis.net/wps/1.0.0/wpsAll.xsd">
  <ows:Identifier>gs:DownloadMap</ows:Identifier>
  <wps:DataInputs>
    <wps:Input>
      <ows:Identifier>bbox</ows:Identifier>
      <wps:Data>
        <wps:BoundingBoxData crs="EPSG:4326">
          <ows:LowerCorner>0.237 40.562</ows:LowerCorner>
          <ows:UpperCorner>14.593 44.55</ows:UpperCorner>
        </wps:BoundingBoxData>
      </wps:Data>
    </wps:Input>
    <wps:Input>
      <ows:Identifier>time</ows:Identifier>
      <wps:Data>
        <wps:LiteralData>2008-10-31T00:00:00.000Z</wps:LiteralData>
      </wps:Data>
    </wps:Input>
    <wps:Input>
      <ows:Identifier>width</ows:Identifier>
      <wps:Data>
        <wps:LiteralData>200</wps:LiteralData>
      </wps:Data>
    </wps:Input>
    <wps:Input>
      <ows:Identifier>height</ows:Identifier>
      <wps:Data>
        <wps:LiteralData>80</wps:LiteralData>
      </wps:Data>
    </wps:Input>
    <wps:Input>
      <ows:Identifier>layer</ows:Identifier>
      <wps:Data>
        <wps:ComplexData xmlns:dwn="http://geoserver.org/wps/download">
          <dwn:Layer>
            <dwn:Name>giantPolygon</dwn:Name>
            <dwn:Parameter key="featureId">giantPolygon.0</dwn:Parameter>
          </dwn:Layer>
        </wps:ComplexData>
      </wps:Data>
    </wps:Input>
    <wps:Input>
      <ows:Identifier>layer</ows:Identifier>
      <wps:Data>
        <wps:ComplexData xmlns:dwn="http://geoserver.org/wps/download">
          <dwn:Layer>
```

```
            <dwn:Name>watertemp</dwn:Name>
          </dwn:Layer>
        </wps:ComplexData>
      </wps:Data>
    </wps:Input>
  </wps:DataInputs>
  <wps:ResponseForm>
    <wps:RawDataOutput mimeType="image/png">
      <ows:Identifier>result</ows:Identifier>
    </wps:RawDataOutput>
  </wps:ResponseForm>
</wps:Execute>
```

For this example the layers could have been a single one, with a "Name" equal to "giantPolygon,watertermp".

Sample DownloadAnimation request

The download animation has all the basic parameters with the following variants/additions:

- time: The time parameter is required and can be provided either as range with periodicity, `start/stop/period`, or as a comma separated list of times,"t1,t2,...,tn"
- fps: Frame per seconds (defaults to one)

A sample animation request can look as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<wps:Execute version="1.0.0" service="WPS"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns=
↪"http://www.opengis.net/wps/1.0.0"
             xmlns:wfs="http://www.opengis.net/wfs" xmlns:wps="http://www.
↪opengis.net/wps/1.0.0"
             xmlns:ows="http://www.opengis.net/ows/1.1" xmlns:gml="http://
↪www.opengis.net/gml"
             xmlns:ogc="http://www.opengis.net/ogc" xmlns:wcs="http://www.
↪opengis.net/wcs/1.1.1"
             xmlns:xlink="http://www.w3.org/1999/xlink"
             xsi:schemaLocation="http://www.opengis.net/wps/1.0.0 http://
↪schemas.opengis.net/wps/1.0.0/wpsAll.xsd">
  <ows:Identifier>gs:DownloadAnimation</ows:Identifier>
  <wps:DataInputs>
    <wps:Input>
      <ows:Identifier>bbox</ows:Identifier>
      <wps:Data>
        <wps:BoundingBoxData crs="EPSG:4326">
          <ows:LowerCorner>-180 -90</ows:LowerCorner>
          <ows:UpperCorner>180 90</ows:UpperCorner>
        </wps:BoundingBoxData>
      </wps:Data>
    </wps:Input>
    <wps:Input>
      <ows:Identifier>decoration</ows:Identifier>
      <wps:Data>
        <wps:LiteralData>formattedTimestamper</wps:LiteralData>
      </wps:Data>
    </wps:Input>
```

```
    <wps:Input>
      <ows:Identifier>time</ows:Identifier>
      <wps:Data>
        <wps:LiteralData>2004-02-01,2004-03-01,2004-04-01,2004-05-01</
↪wps:LiteralData>
      </wps:Data>
    </wps:Input>
    <wps:Input>
      <ows:Identifier>width</ows:Identifier>
      <wps:Data>
        <wps:LiteralData>271</wps:LiteralData>
      </wps:Data>
    </wps:Input>
    <wps:Input>
      <ows:Identifier>height</ows:Identifier>
      <wps:Data>
        <wps:LiteralData>136</wps:LiteralData>
      </wps:Data>
    </wps:Input>
    <wps:Input>
      <ows:Identifier>fps</ows:Identifier>
      <wps:Data>
        <wps:LiteralData>0.5</wps:LiteralData>
      </wps:Data>
    </wps:Input>
    <wps:Input>
      <ows:Identifier>layer</ows:Identifier>
      <wps:Data>
        <wps:ComplexData xmlns:dwn="http://geoserver.org/wps/download">
          <dwn:Layer>
            <dwn:Name>sf:bmtime</dwn:Name>
          </dwn:Layer>
        </wps:ComplexData>
      </wps:Data>
    </wps:Input>
  </wps:DataInputs>
  <wps:ResponseForm>
    <wps:RawDataOutput mimeType="video/mp4">
      <ows:Identifier>result</ows:Identifier>
    </wps:RawDataOutput>
  </wps:ResponseForm>
</wps:Execute>
```

The `formattedTimestamper` decoration ensures the frame time is included in the output animation, and looks as follows:

```
<layout>
  <decoration type="text" affinity="bottom,right" offset="6,6" size="auto">
    <option name="message"><![CDATA[
<#setting datetime_format="yyyy-MM-dd'T'HH:mm:ss.SSSX">
<#setting locale="en_US">
<#if time??>
${time?datetime?string["dd-MM-yyyy"]}
</#if>]]></option>
    <option name="font-family" value="Bitstream Vera Sans"/>
    <option name="font-size" value="12"/>
    <option name="halo-radius" value="2"/>
  </decoration>
```

```
</layout>
```

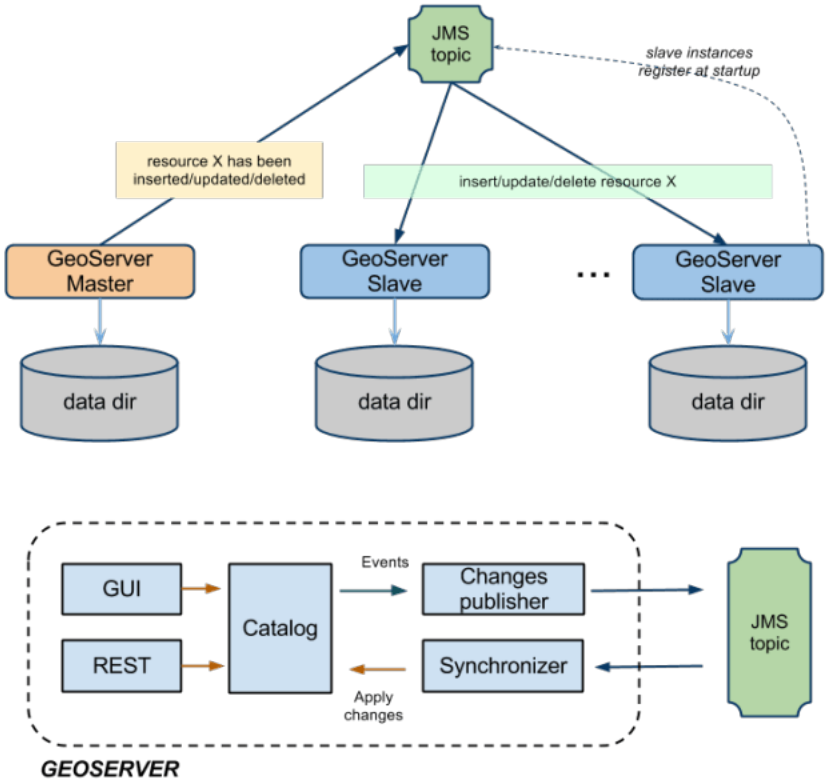
Decoration Layout

The `decoration` parameter specifies the file name (without extension) of the layout to be used to decorate the map.
The layout is a list of decorators that should draw on top of the requested image.
The decorators draw on the image one after the other, so the order of the decorators in the layout file is important: the first decorator output will appear under the others.
Decorators are described in detail in the *WMS Decorations* section.

16.13 JMS based Clustering

16.13.1 Introduction

The are several approaches to implement a clustered deployment with GeoServer, based on different mixes of data directory sharing plus configuration reload. The JMS based clustering module architecture is depicted in the following diagram:



This module implements a robust Master/Slave approach which leverages on a Message Oriented Middleware (MOM) where:

- The Masters accept changes to the internal configuration, persist them on their own data directory but also forward them to the Slaves via the MOM
- The Slaves should not be used to change their configuration from either REST or the User Interface, since are configured to inject configuration changes disseminated by the Master(s) via the MOM
- The MOM is used to make the Master and the Slave exchange messages in a durable fashion
- Each Slave has its own data directory and it is responsible for keeping it aligned with the Master's one. In case a Slave goes down when it goes up again he might receive a bunch of JMS messages to align its configuration to the Master's one.
- A Node can be both Master and Slave at the same time, this means that we don't have a single point of failure, i.e. the Master itself

Summarizing, the Master as well as each Slave use a private data directory, Slaves receive changes from the Master, which is the only one where configuration changes are allowed, via JMS messages. Such messages transport GeoServer configuration objects that the Slaves inject directly in their own in-memory configuration and then persist on disk on their data directory, completely removing the need for a configuration reload for each configuration change.

To make things simpler, in the default configuration the MOM is actually embedded and running inside each GeoServer, using multicast to find its peers and thus allowing for a clustering installation without the need to have a separate MOM deploy.

16.13.2 Description

The GeoServer Master/slave integration is implemented using JMS, Spring and a MOM (Message Oriented Middleware), in particular ActiveMQ. The schema in Illustration represents a complete high level design of Master/Slave platform. It is composed by 3 distinct actors:

1. GeoServer Masters
2. GeoServer Slaves
3. The MOM (ActiveMQ)

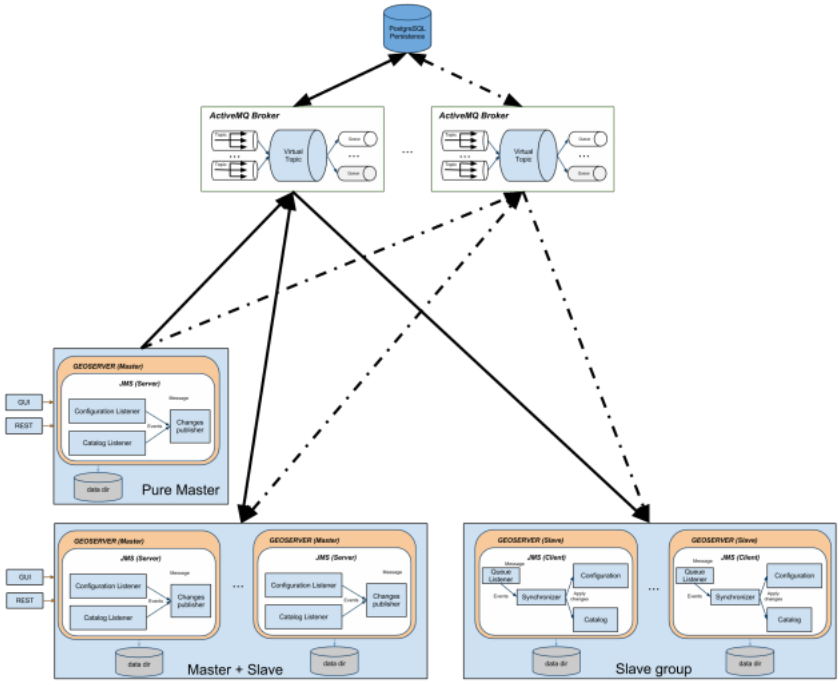
This structure allows to have:

1. Queue fail-over components (using MOM).
2. Slaves down are automatically handled using durable topic (which will store message to re-synch changes happens if one of the slaves is down when the message was made available)
3. Master down will not affect any slave synchronization process.

This full blown deployment is composed by:

- A pure Master GeoServer(s), this instance can only send events to the topic. It cannot act as a slave
- A set of GeoServer which can work as both Master and Slave. These instances can send and receive messages to/from the topic. They can work as Masters (sending message to other subscribers) as well as Slaves (these instances are also subscribers of the topic).
- A set of pure Slaves GeoServer instances whic can only receive messages from the topic.
- A set of MOM brokers so that each GeoServer instance is configured with a set of available brokers (failover). Each broker use the shared database as persistence. Doing so if a broker fails for some reason, messages can still be written and read from the shared database.

All the produced code is based on spring-jms to ensure portability amongst different MOM, but if you look at the schema, we are also leveraging ActiveMQ VirtualTopics to get dinamic routing (you can dinamically attach masters and slaves).



The VirtualTopics feature has also other advantages explained here: http://activemq.apache.org/virtual-destinations.html

As said above, in the default configuration the MOM runs inside GeoServer itself, simplifying the topology and the setup effort.

Installation

To install the jms cluster modules into an existing GeoServer refer to the *Installation of the JMS Cluster modules* page

16.13.3 How to configure GeoServer Instances

The configuration for the GeoServer is very simple and can be performed using the provided GUI or modifying the `cluster.properties` file which is stored into the `GEOSERVER_DATA_DIR` under the `cluster` folder (e.g. `${GEOSERVER_DATA_DIR}/cluster`).

To override the default destination of this configuration file you have to setup the **CLUSTER_CONFIG_DIR** variable defining the destination folder of the **cluster.properties** file. This is *mandatory* when you want to share the same GEOSERVER_DATA_DIR, but not required if you are giving each GeoServer its own data directory.

In case of shared data directory, it will be necessary to add a different system variable to each JVM running a GeoServer, e.g.:

- `-DCLUSTER_CONFIG_DIR=/var/geoserver/clusterConfig/1` to the JVM running the first node
- `-DCLUSTER_CONFIG_DIR=/var/geoserver/clusterConfig/2` to the JVM running the second node
- and so on

If the directories are missing, GeoServer will create them and provide a initial `cluster.properties` with reasonable defaults.

16.13.4 Instance name

The instance.name is used to distinguish from which GeoServer instance the message is coming from, so each GeoServer instance should use a different, unique (for a single cluster) name.

16.13.5 Broker URL

The broker.url field is used to instruct the internal JMS machinery where to publish messages to (master GeoServer installation) or where to consume messages from (slave GeoServer installation). Many options are available for configuring the connection between the GeoServer instance and the JMS broker, for a complete list, please, check http://activemq.apache.org/configuring-transports.html. In case when (recommended) failover set up is put in place multiple broker URLs can be used. See http://activemq.apache.org/failover-transport-reference.html for more information about how to configure that. Note GeoServer will not complete the start-up phase until the target broker is correctly activated and reachable.

16.13.6 Limitations and future extensions

Data

NO DATA IS SENT THROUGH THE JMS CHANNEL The clustering solution we have put in place is specific for managing the GeoServer internal configuration, no data is transferred between master and slaves. For that purpose use external mechanisms (ref. [GeoServer REST]). In principle this is not a limitation per se since usually in a clustered environment data is stored in shared locations outside the data directory. With our solution this is a requirement since each slave will have its own private data directory.

Things to avoid

- **NEVER RELOAD THE GEOSERVER CATALOG ON A MASTER**: Each master instance should never call the catalog reload since this propagates the creation of all the resources, styles, etc to all the connected slaves.
- **NEVER CHANGE CONFIGURATION USING A PURE SLAVE**: This will make the configuration of the specific slave out of synch with the others.

16.13.7 Refs:

Installation of the JMS Cluster modules

To install the JMS Cluster modules you simply have to:

- Download the `geoserver-jms-cluster-<version>.zip` file from the nightly builds, community section
- Stop GeoServer
- Unpack the zip file in `webapps/geoserver/WEB-lib`
- Start again GeoServer

: .. warning:

```
In GeoServer versions 2.10.4, 2.11.1 and 2.12-beta default topic name was renamed␣
↪from ``VirtualTopic.>``, which has a special meaning in ActiveMQ, to ``VirtualTopic.
↪geoserver``. When upgrading to one of this versions or above the virtual topic name␣
↪will be automatically updated. Note that only GeoServer instances that use the same␣
↪topic name will be synchronized.
```

As a recommendation, for the first tests try to run the cluster module on a cluster of GeoServer all having their own data directory.

If you want to use the clustering extension while sharing the same data dir that's also possible, but you'll have to remember to use the `CLUSTER_CONFIG_DIR` system variable, and set it to a different folder for each instance, e.g.:

- set `-DGEOSERVER_CONFIG_DIR=/path/to/cluster/config/dir/1` on the first node,
- set `-DGEOSERVER_CONFIG_DIR=/path/to/cluster/config/dir/2` on the second node
- and so on

The directories do not need to exist, GeoServer will create and populate them on startup automatically.

HOWTO configure ActiveMQ broker

Deploy the produced activemqBroker.war in your tomcat instance and check the extracted webapp. You may locate a file called activemq-jmx.properties which will help you to configure your instance with the most important paramenters. Anyhow it is only an example and we encourage you to also check the ApplicationContext.xml file deployed to activemq/WEB-INF/classes/ApplicationContext.xml which is the complete configuration:

```
...
<!-- The transport connectors expose ActiveMQ over a given protocol to
      clients and other brokers.
      For more information, see: http://activemq.apache.org/configuring-transports.
↪html -->
<transportConnectors>
        <transportConnector name="openwire" uri="tcp://192.168.1.XXX:61616" />
</transportConnectors>
...
```

Persistence Configuration

It is possible to enable persistence for messages that cannot be delivered right away (e.g. all consumers are down). Detailed information can be found here, we are simply going to provide basic information on how to achieve that. To configure the persistence for the messages to deliver you need to setup the <persistenceAdapter> node in the same file as above and then configure a proper datasource in your DBMS of choice.

```
...
<persistenceAdapter>
<!-- <kahaDB directory="${activemq.base}/data/kahadb"/> -->
  <jdbcPersistenceAdapter dataDirectory="activemq-data"
        dataSource="#postgres-ds" lockKeepAlivePeriod="0"/>
</persistenceAdapter>
...
```

In the above section we defined a jdbcPersistenceAdapter connected to a dataSource called #postgres-ds that forces the broker to use PostgreSQL for persisting its messages when the delivery cannot be guaranteed (e.g. a slave goes down unexpectedly). You now need to configure your own datasource as specified in the following section which are specific for different DBMS.

Oracle datasource

To configure the broker to use an oracle database as datasource you need to uncomment and modify the following peace into the ApplicationContext.xml file:

```
...
<bean id="oracle-ds" class="org.apache.commons.dbcp.BasicDataSource" destroy-method=
↪"close">
  <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver"/>
  <property name="url" value="jdbc:oracle:thin:@localhost:1521:AMQDB"/>
  <property name="username" value="oracle"/>
  <property name="password" value=" oracle "/>
  <property name="poolPreparedStatements" value="true"/>
</bean>
...
```

In addition, you need to make sure that the jar containing the driver for Oracle is correctly deployed inside the WEB-INF/lib for the activemq war file. At the same time the database referred in provided instructions as well as the user must be already present.

Postgres datasource

Configuring PostgreSQL as the datasource to use for the persistence of the messages for the ActiveMQ broker follows the same pattern as above. See below for some examples.

```
...
<bean id="postgres-ds" class="org.postgresql.ds.PGPoolingDataSource">
        <property name="serverName" value="192.168.1.XXX"/>
        <property name="databaseName" value="activemq"/>
        <property name="portNumber" value="5432"/>
        <property name="user" value="postgres"/>
        <property name="password" value="postgres"/>
        <property name="dataSourceName" value="postgres"/>
        <property name="initialConnections" value="15"/>
        <property name="maxConnections" value="30"/>
</bean>
...
```

Note: The above ApplicationContext.xml file contains some unused sections which are intentionally commented out to show different types of configurations [Ref. ActiveMQ].

Kaha datasource (Embedded database)

Besides using server DBMS as indicated above we can use embedded database for simpler uses cases of demoing since this usually largely simplify the configuration. At this link all the information needed for

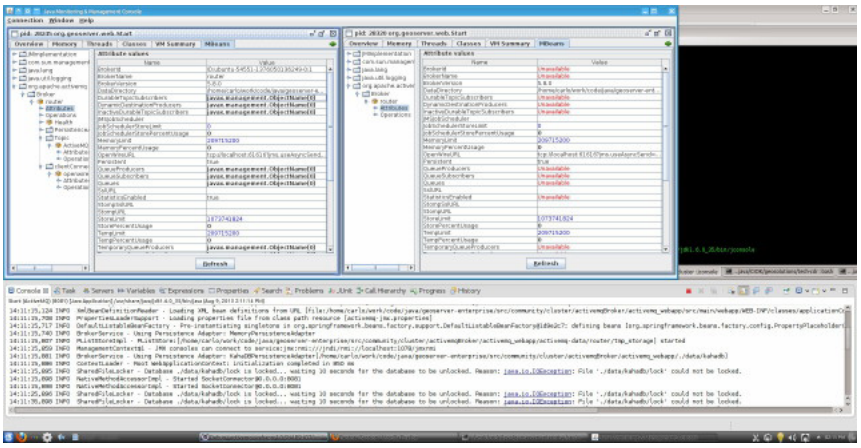
achieving this result can be found; basically we need to uncomment the related datasource and then reference it from the persistenceAdapter.

Control instances using JMX

Be sure to edit the activemq-jmx.properties (or via the environment variables) setting different JMX ports for different broker instances. Deploy as explained the instances into 2 different webapplication container (f.e. Tomcat) and start both application (on different port f.e. 8081 and 8082). Now run jconsole to connect to the brokers via JMX:

${JAVA_HOME}/bin/jconsole

After you connect to the brokers you may see something like this:



You may look at the console, as you can see the 2nd instance of the broker cannot take the look on the file (the example uses KahaDB); this is also visible in the JMX console into the widhow on the right side.

If now you select the 'operation' (on the left side window) you will see:

Using that console we are able to perform many operation, so to simulate a broker down we try to click on the 'stop()' button.

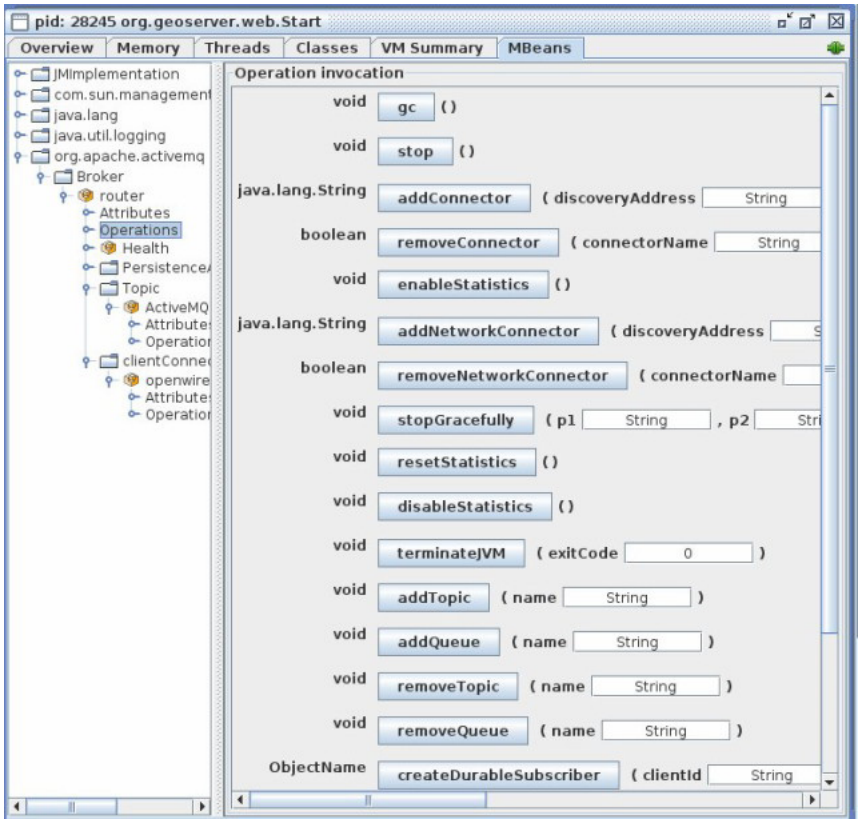
Doing so, the first broker instance will stop and the JMX connection will be closed, and the second instance (on the right side) will keep the control of the DB.

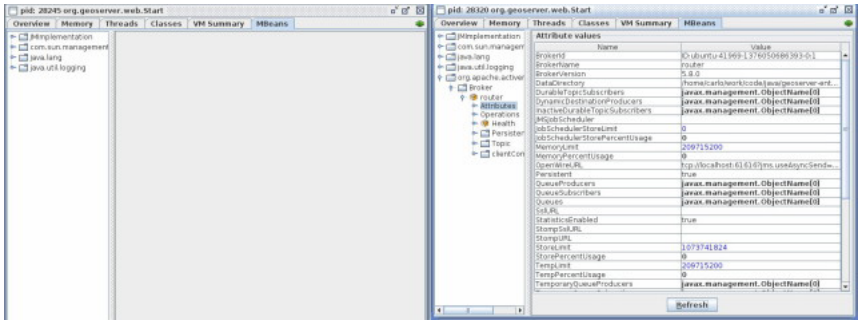
JDBC Master Slave

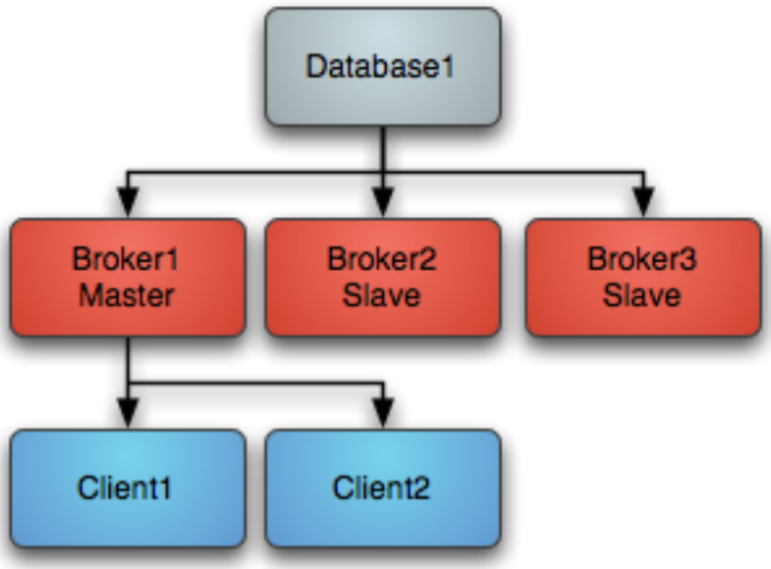
If you are using pure JDBC and not using the high performance journal then you are generally relying on your database as your single point of failure and persistence engine. If you do not have really high performance requirements this approach can make a lot of sense as you have a single persistence engine to backup and manage etc.

Startup

When using just JDBC as the data source you can use a Master Slave approach, running as many brokers as you wish as this diagram shows. On startup one master grabs an exclusive lock in the broker database - all other brokers are slaves and pause waiting for the exclusive lock.


pid: 28245 org.geoserver.web.Start
Overview
Memory
Threads
Classes
VM Summary
MBeans
JMImplementation
com.sun.management
java.lang
java.util.logging
org.apache.activemq
Broker
router
Attributes
Operations
Health
Topic
Operation invocation
void gc ()
void stop ()
java.lang.String addConnector (discoveryAddress String
boolean removeConnector (connectorName String
void enableStatistics ()
java.lang.String addNetworkConnector (discoveryAddress
boolean removeNetworkConnector (connectorName
void stopGracefully (p1 String , p2
void resetStatistics ()
void disableStatistics ()
void terminateJVM (exitCode 0)
void addTopic (name String)
void addQueue (name String)
void removeTopic (name String)
void removeQueue (name String)
ObjectName createDurableSubscriber (clientId String


Attribute values
Refresh



Clients should be using the Failover Transport to connect to the available brokers. e.g. using a URL something like the following failover:(tcp://broker1:61616,tcp://broker2:61616,tcp://broker3:61616) Only the master broker starts up its transport connectors and so the clients can only connect to the master.

Master failure

If the master looses connection to the database or looses the exclusive lock then it immediately shuts down. If a master shuts down or fails, one of the other slaves will grab the lock and so the topology switches to the following diagram

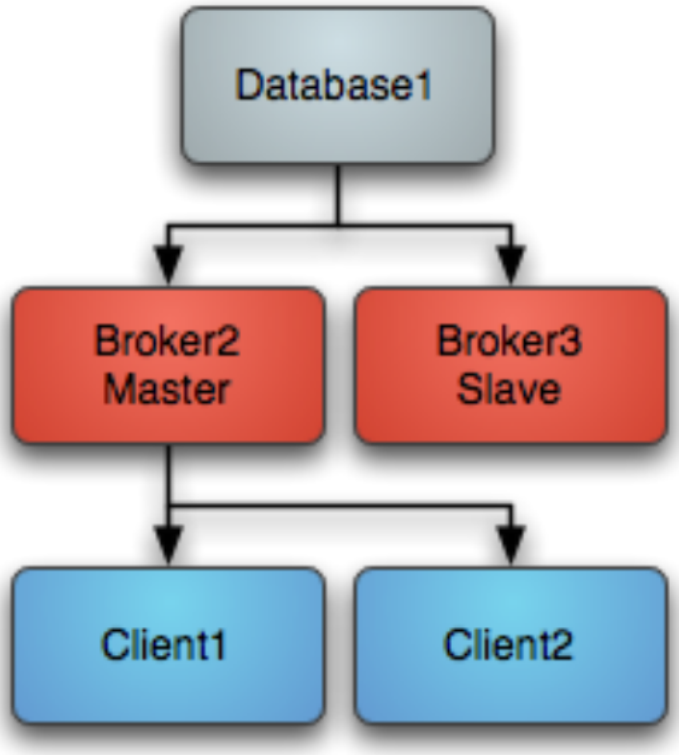
One of the other other slaves immediately grabs the exclusive lock on the database to them commences becoming the master, starting all of its transport connectors. Clients loose connection to the stopped master and then the failover transport tries to connect to the available brokers - of which the only one available is the new master. Master restart At any time you can restart other brokers which join the cluster and start as slaves waiting to become a master if the master is shutdown or a failure occurs. So the following topology is created after a restart of an old master...

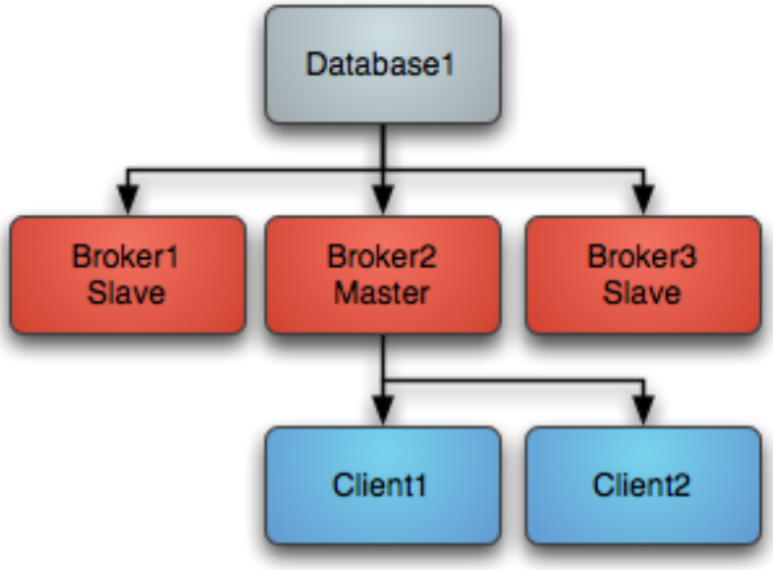
Configuring JDBC Master Slave

By default if you use the <jdbcPersistenceAdapter/> to avoid the high performance journal you will be using JDBC Master Slave by default. You just need to run more than one broker and point the client side URIs to them to get master/slave. This works because they both try an acquire an exclusive lock on a shared table in the database and only one will succeed.

The following example shows how to configure the ActiveMQ broker in JDBC Master Slave mode

```
<beans>

  <!-- Allows us to use system properties as variables in this configuration file -->
  <bean class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer
↪"/>
```


Database1
Broker2
Master
Broker3
Slave
Client1
Client2


Database1
Broker1
Slave
Broker2
Master
Broker3
Slave
Client1
Client2

```
  <broker xmlns="http://activemq.apache.org/schema/core">

    <destinationPolicy>
      <policyMap><policyEntries>

          <policyEntry topic="FOO.>">
            <dispatchPolicy>
              <strictOrderDispatchPolicy />
            </dispatchPolicy>
            <subscriptionRecoveryPolicy>
              <lastImageSubscriptionRecoveryPolicy />
            </subscriptionRecoveryPolicy>
          </policyEntry>

      </policyEntries></policyMap>
    </destinationPolicy>


    <persistenceAdapter>
        <jdbcPersistenceAdapter dataDirectory="${activemq.base}/activemq-data"/>

        <!--
        <jdbcPersistenceAdapter dataDirectory="activemq-data" dataSource="#oracle-ds"/
↪>
        -->
    </persistenceAdapter>

    <transportConnectors>
      <transportConnector name="default" uri="tcp://localhost:61616"/>
    </transportConnectors>

  </broker>

  <!--  This xbean configuration file supports all the standard spring xml␣
↪configuration options -->

  <!-- Postgres DataSource Sample Setup -->
  <!--
  <bean id="postgres-ds" class="org.postgresql.ds.PGPoolingDataSource">
    <property name="serverName" value="localhost"/>
    <property name="databaseName" value="activemq"/>
    <property name="portNumber" value="0"/>
    <property name="user" value="activemq"/>
    <property name="password" value="activemq"/>
    <property name="dataSourceName" value="postgres"/>
    <property name="initialConnections" value="1"/>
    <property name="maxConnections" value="10"/>
  </bean>
  -->

  <!-- MySql DataSource Sample Setup -->
  <!--
  <bean id="mysql-ds" class="org.apache.commons.dbcp.BasicDataSource" destroy-method=
↪"close">
    <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
    <property name="url" value="jdbc:mysql://localhost/activemq?relaxAutoCommit=true"/
↪>
    <property name="username" value="activemq"/>
```

```
    <property name="password" value="activemq"/>
    <property name="poolPreparedStatements" value="true"/>
  </bean>
  -->

  <!-- Oracle DataSource Sample Setup -->
  <!--
  <bean id="oracle-ds" class="org.apache.commons.dbcp.BasicDataSource" destroy-method=
↪"close">
    <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver"/>
    <property name="url" value="jdbc:oracle:thin:@localhost:1521:AMQDB"/>
    <property name="username" value="scott"/>
    <property name="password" value="tiger"/>
    <property name="poolPreparedStatements" value="true"/>
  </bean>
  -->

  <!-- Embedded Derby DataSource Sample Setup -->
  <!--
  <bean id="derby-ds" class="org.apache.derby.jdbc.EmbeddedDataSource">
    <property name="databaseName" value="derbydb"/>
    <property name="createDatabase" value="create"/>
  </bean>
  -->

</beans>
```

Shared File System Master Slave

Basically you can run as many brokers as you wish from the same shared file system directory. The first broker to grab the exclusive lock on the file is the master broker. If that broker dies and releases the lock then another broker takes over. The slave brokers sit in a loop trying to grab the lock from the master broker. The following example shows how to configure a broker for Shared File System Master Slave where /sharedFileSystem is some directory on a shared file system. It is just a case of configuring a file based store to use a shared directory.

```
<persistenceAdapter>
  <kahaDB directory="/sharedFileSystem/sharedBrokerData"/>
</persistenceAdapter>
```

or:

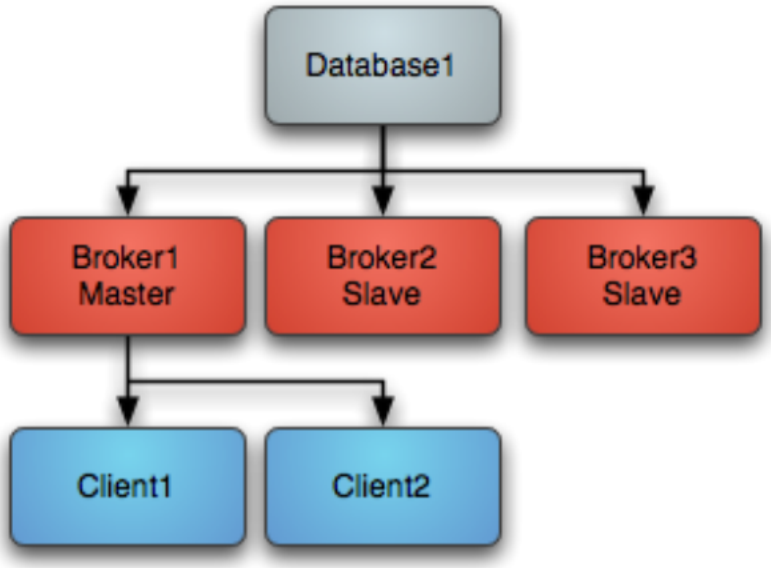
```
<persistenceAdapter>
  <levelDB directory="/sharedFileSystem/sharedBrokerData"/>
</persistenceAdapter>
```

or:

```
<persistenceAdapter>
  <amqPersistenceAdapter directory="/sharedFileSystem/sharedBrokerData"/>
</persistenceAdapter>
```

Startup

On startup one master grabs an exclusive lock on the broker file directory - all other brokers are slaves and pause waiting for the exclusive lock.



Clients should be using the Failover Transport to connect to the available brokers. e.g. using a URL something like the following

```
failover:(tcp://broker1:61616,tcp://broker2:61616,tcp://broker3:61616)
```

Only the master broker starts up its transport connectors and so the clients can only connect to the master.

Master failure

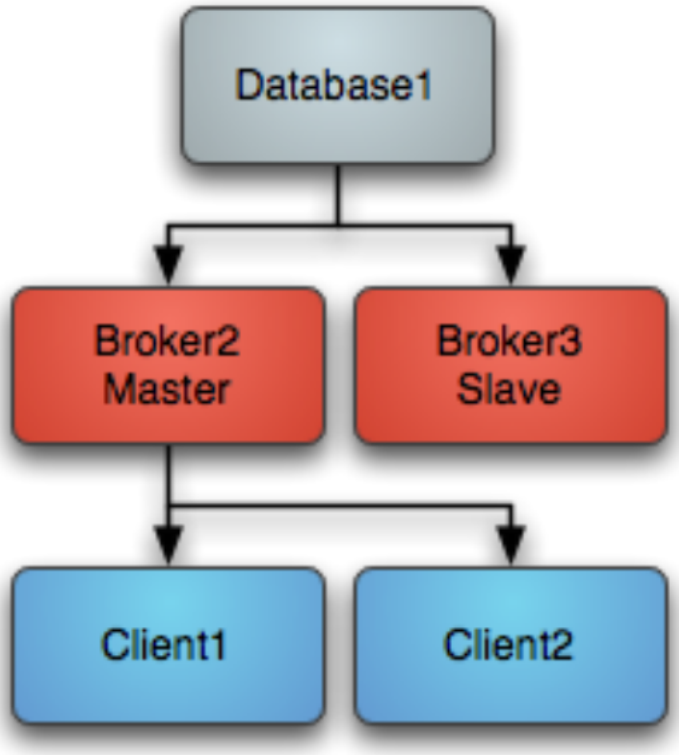
If the master looses the exclusive lock then it immediately shuts down. If a master shuts down or fails, one of the other slaves will grab the lock and so the topology switches to the following diagram

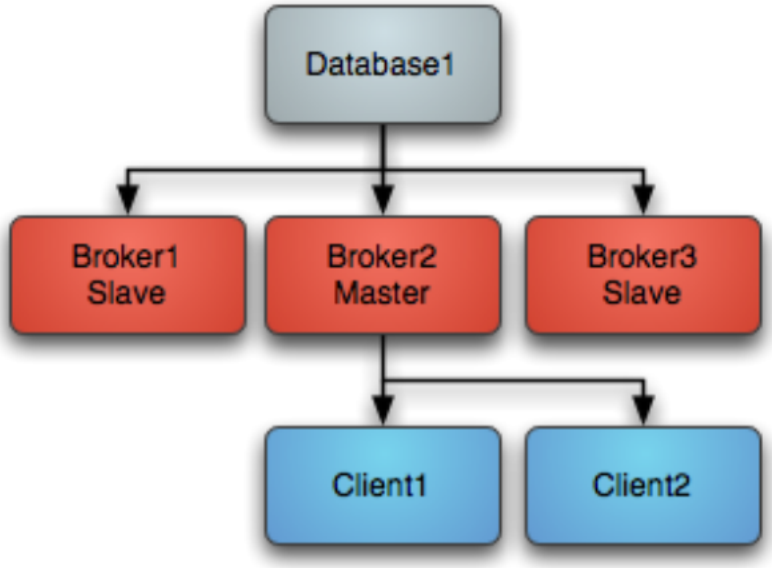
One of the other other slaves immediately grabs the exclusive lock on the file system to them commences becoming the master, starting all of its transport connectors. Clients loose connection to the stopped master and then the failover transport tries to connect to the available brokers - of which the only one available is the new master.

Master restart

At any time you can restart other brokers which join the cluster and start as slaves waiting to become a master if the master is shutdown or a failure occurs. So the following topology is created after a restart of an old master...

Note: If you have a SAN or shared file system it can be used to provide high availability such that if a broker is killed, another broker can take over immediately.


Database1
Broker2
Master
Broker3
Slave
Client1
Client2


Database1
Broker1
Slave
Broker2
Master
Broker3
Slave
Client1
Client2

Ensure your shared file locks work

Note that the requirements of this failover system are a distributed file system like a SAN for which exclusive file locks work reliably. If you do not have such a thing available then consider using MasterSlave instead which implements something similar but working on commodity hardware using local file systems which ActiveMQ does the replication.

OCFS2 Warning

Was testing using OCFS2 and both brokers thought they had the master lock - this is because "OCFS2 only supports locking with 'fcntl' and not 'lockf and flock', therefore mutex file locking from Java isn't supported."

From http://sources.redhat.com/cluster/faq.html#gfs_vs_ocfs2 :

OCFS2: No cluster-aware flock or POSIX locks

GFS: fully supports Cluster-wide flocks and POSIX locks and is supported.

NFSv3 Warning

In the event of an abnormal NFSv3 client termination (i.e., the ActiveMQ master broker), the NFSv3 server will not timeout the lock that is held by that client. This effectively renders the ActiveMQ data directory inaccessible because the ActiveMQ slave broker can't acquire the lock and therefore cannot start up. The only solution to this predicament with NFSv3 is to reboot all ActiveMQ instances to reset everything.

Use of NFSv4 is another solution because it's design includes timeouts for locks. When using NFSv4 and the client holding the lock experiences an abnormal termination, by design, the lock is released after 30 seconds, allowing another client to grab the lock. For more information about this, see this blog entry.

16.13.8 Bibliography:

[JMS specs] Sun microsystens - Java Message Service - Version 1.1 April 12, 2002

[JMS] Snyder Bosanac Davies - ActiveMQ in action - Manning

[GeoServer] http://docs.geoserver.org/

[GeoServer REST] http://docs.geoserver.org/latest/en/user/rest/index.html#rest

[ActiveMQ] http://activemq.apache.org/

16.14 SOLR data store

SOLR is a popular search platform based on Apache Lucene project. Its major features include powerful full-text search, hit highlighting, faceted search, near real-time indexing, dynamic clustering, database integration, rich document (e.g., Word, PDF) handling, and most importantly for the GeoServer integration, geospatial search.

The latest versions of SOLR can host most basic types of geometries (points, lines and polygons) as WKT and index them with a spatial index.

Note: GeoServer does not come built-in with support for SOLR; it must be installed through this community module.

The GeoServer SOLR extension has been tested with SOLR version 4.8, 4.9, and 4.10.

The extension supports all WKT geometry types (all linear types, point, lines and polygons, SQL/MMcurves are not supported), plus "bounding box" (available starting SOLR 4.10). It does not support the `solr.LatLonType` type yet.

The following pages shows how to use the SOLR data store.

16.14.1 SOLR layer configuration

Mapping documents to layers

SOLR indexes almost free form documents, the SOLR instance has a collection of fields, and each document can contain any field, in any combination. On the other side, GeoServer organizes data in fixed structure feature types, and exposes data in separate layers. This leaves the question of how documents in the index should be organized into layers.

By default the store exposes a single layer, normally named after the SOLR collection the store is connected to, by publishing it one can decide which fields to include, and eventually add a filter to select which attributes it will contain.

This single layer can be published multiple times, giving each published layer a different name, set of selected attributes, and a different filter to select the documents contained in the layer.

Installing the SOLR extension

1. Download the SOLR extension from the nightly GeoServer community module builds.

   Warning: Make sure to match the version of the extension to the version of the GeoServer instance.

2. If GeoServer is running, stop it.
3. Extract the contents of the archive into the `WEB-INF/lib` directory of the GeoServer installation.
4. Restart GeoServer, the SOLR data store should show up as an option when going through the new store creation workflow.

Connecting to a SOLR server

Once the extension is properly installed `SOLR` will show up as an option when creating a new data store.

Configuring a SOLR data store

`solr_url`	Provide a link to the SOLR server that provides the documents

Once the parameters are entered and confirmed, GeoServer will contact the SOLR server and fetch a list of layer names and fill the layer chooser page accordingly:

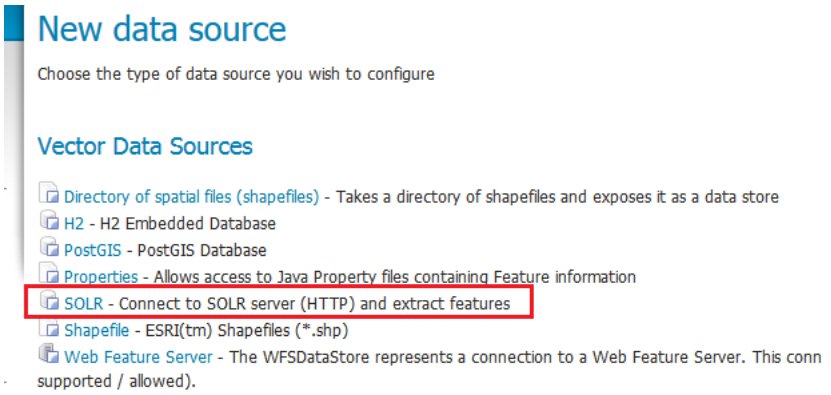


Fig. 16.5: *SOLR in the list of vector data sources*

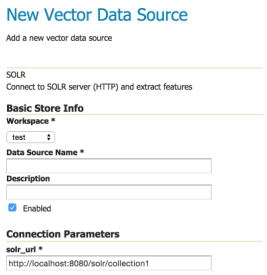


Fig. 16.6: *Configuring a SOLR data store*

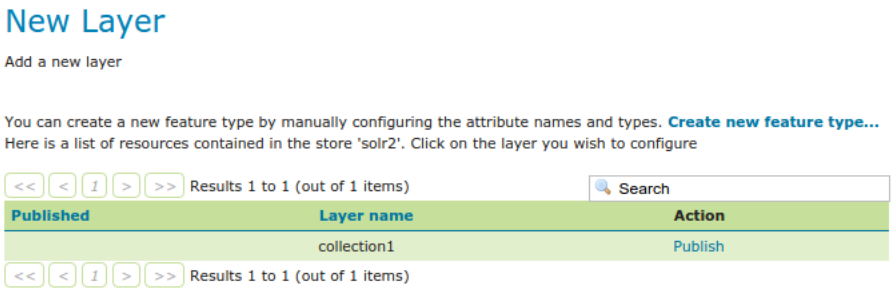


Fig. 16.7: *List of layers available in the SOLR server*

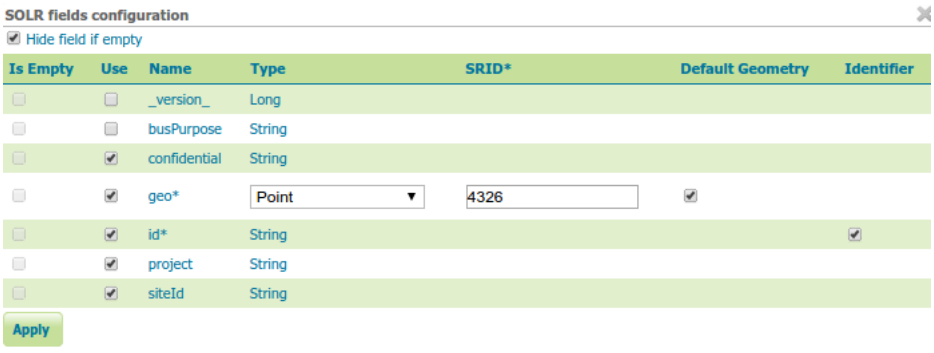


Fig. 16.8: *The layer field list configuration*

Configuring a new SOLR base layer

Once the layer name is chosen, the usual layer configuration panel will appear, with a pop-up showing in a table the fields available:

`Is empty`	Read only fields, checked if the field has no values in the documents associated to this layer
`Use`	Used to select the fields that will make up this layer features
`Name`	Name of the field
`Type`	Type of the field, as derived from the SOLR schema. For geometry types, you have the option to provide a more specific data type
`SRID`	Native spatial reference ID of the geometries
`Default geometry`	Indicates if the geometry field is the default one. Useful if the documents contain more than one geometry field, as SLDs and spatial filters will hit the default geometry field unless otherwise specified
`Identifier`	Check if the field can be used as the feature identifier

By default the list will contain only the fields that have at least one non null value in the documents associated to the layer, but it is possible to get the full list by un-checking the "Hide field if empty" check-box:

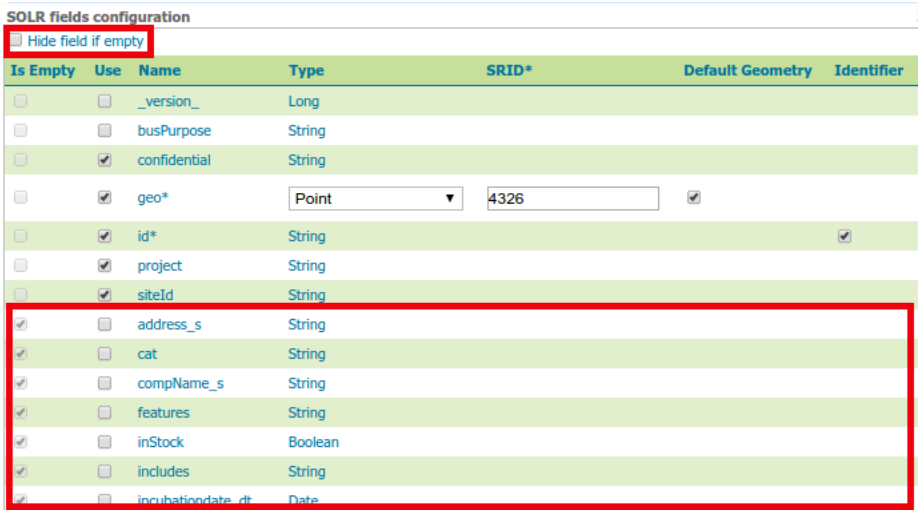


Fig. 16.9: *Showing all fields available in SOLR*

Once the table is filled with the all the required parameters, press the "Apply" button to confirm and go back to the main layer configuration panel. Should the choice of fields be modified, you can click the "Configure SOLR fields" just below the "Feature Type Details" panel.

The rest of the layer configuration works as normal, once all the fields are provided you'll be able to save and use the layer in WMS and WFS.

Warning: In order to compute the bounding box GeoServer will have to fetch all the geometries making up the layer out of SOLR, this operation might take some time, you're advised to manually entered the native bounding box when configuring a layer out of a large document set

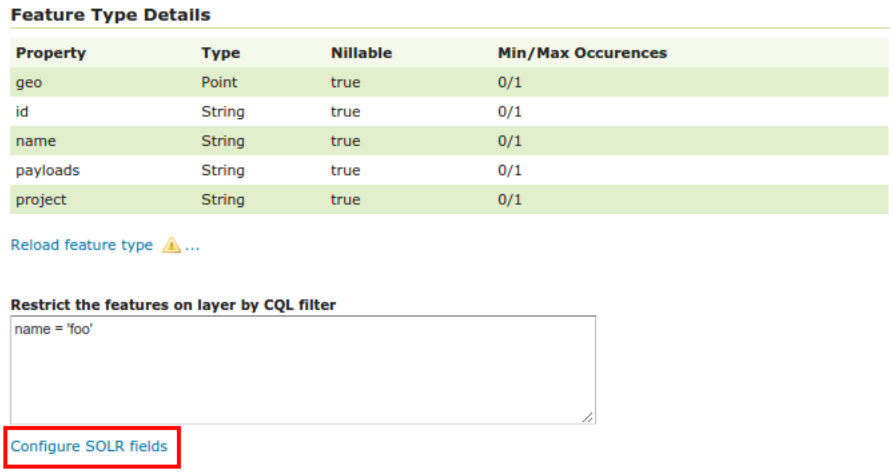


Fig. 16.10: *Going back to the field list editor*

Custom `q` and `fq` parameters

The SOLR store will translate most OGC filters, as specified in SLD, CQL Filter or OGC filter, down into the SOLR engine for native filtering, using the `fq` parameter. However, in some occasions you might need to specify manually either `q` or `fq`, to leverage some native SOLR filtering ability that cannot be expressed via OGC filters.

This can be done by specifying those as `viewparams`, pretty much like in parametric sql views atop relational databases.

For example, the following URL:

```
http://localhost:8080/geoserver/nurc/wms?service=WMS&version=1.1.0&request=GetMap
     &layers=nurc:active&styles=geo2&bbox=0.0,0.0,24.0,44.0&width=279&height=512
     &srs=EPSG:4326&format=application/openlayers
     &viewparams=fq:security_ss:WEP
```

Will send down to SOLR a query looking like:

```
omitHeader=true&fl=geo,id&q=*:*&rows=2147483647&sort=id asc
&fq=status_s:active AND geo:"Intersects(POLYGON ((-0.125 -0.5333333333333333, -0.125
↪44.53333333333333,
24.125 44.53333333333333, 24.125 -0.5333333333333333, -0.125 -0.5333333333333333)))"
&fq=security_ss:WEP&cursorMark=*
```

You can notice that:

- Only the columns needed for the display (in this case, a single geometry) are retrieved
- The bbox and layer identification filters are specified in the first `fq`
- The custom `fq` is passed as a second `fq` parameter (SOLR will treat it as being and-ed with the previuos one)

16.14.2 Loading spatial data into SOLR

This section provides a simple example on how to convert and load a shapefile into a SOLR instance. For more advanced needs and details about spatial support in SOLR consult the SOLR documentation, making

sure to read the one associated to the version at hand (spatial support is still rapidly evolving).

The current example has been developed and tested using GDAL 1.11 and SOLR 4.8, different versions of the tools and server might require a different syntax for upload.

The SOLR instance is supposed to have the following definitions in its schema:

```
<field name="geo" type="location_rpt" indexed="true" stored="true" multiValued="true"␣
↪/>
<dynamicField name="*_i"  type="int"    indexed="true"  stored="true"/>
<dynamicField name="*_s"  type="string"  indexed="true"  stored="true" />
```

The above defines "geo" as explicit fields, leaving the other types to dynamic field interpretation.

The SpatialRecursivePrefixTreeFieldType accepts geometries as WKT, so as a preparation for the import we are going to turn a shapefile into a CSV file with WKT syntax for the geometry. Let's also remember that SOLR needs a unique id field for the records, and that the coordinates are supposed to be in WGS84. The shapefile in question is instead in UTM, has a linestring geometry, and some fields, cat,id and label.

The following command translates the shapefile in CSV (the command should be typed in a single line, it has been split over multiple lines for ease of reading):

```
ogr2ogr  -f CSV
         -sql 'select FID as id, cat as cat_i, label as label_s,
              "roads" as layer FROM roads'
         -lco geometry=AS_WKT -s_srs "EPSG:26713" -t_srs "EPSG:4326"
         /tmp/roads.csv roads.shp
```

Some observations:

- The SQL is used mostly to include the special FID field into the results (a unique field is required)
- The reprojection is performed to ensure the output geometries are in WGS84
- The `layer_s` dynamic field is added to

This will generate a CSV file looking as follows:

```
WKT,id,cat_i,label_s,layer
"LINESTRING (-103.763291353072518 44.375039982911382,-103.763393874038698 44.
↪375282535746727,-103.764152625689903 44.376816068582023,-103.763893508430911 44.
↪377653708326527,-103.76287152579593 44.378473197876396,-103.762075892308829 44.
↪379009292692757,-103.76203441159079 44.379195585236509,-103.762124217456204 44.
↪379295262047272,-103.762168141872152 44.379399997909999,-103.762326134985983 44.
↪379527769244149,-103.763328403265064 44.380245486928708,-103.7640011871363465 44.
↪381295133519728,-103.76411460103661 44.381526706124056,-103.764953940327757 44.
↪382396618315049,-103.765097289111338 44.382919576408355,-103.765147974157941 44.
↪383073790503197,-103.76593766187851 44.384162856249255,-103.765899236602976 44.
↪384607239970421,-103.765854384388703 44.384597320206453)",0,5,unimproved road,roads
"LINESTRING (-103.762930948900078 44.385847721442218,-103.763012156628747 44.
↪386002223293282,-103.763510654805799 44.386297912655408,-103.763869052966967 44.
↪386746022746649,-103.763971116268394 44.387444295314552,-103.764244098825387 44.
↪387545690358827,-103.764264649212294 44.387677659170357,-103.764160551326043 44.
↪387951214930865,-103.764540576800869 44.388042632912118,-103.764851624437995 44.
↪388149874425885,-103.764841258550391 44.388303515682807,-103.76484332449354 44.
↪388616502755184,-103.765188923261391 44.388927221995502,-103.765110961905023 44.
↪389448103450221,-103.765245311197177 44.389619574129583,-103.765545516097987 44.
↪389907903843323,-103.765765403056434 44.390420596862072,-103.766285436779711 44.
↪391655378673697,-103.766354640463163 44.39205684519964,-103.76638734105434 44.
↪392364628456725,-103.766410556756725 44.392776645318136,-103.765934443919321 44.
↪393365174368313,-103.766220869020188 44.393571013181166,-103.766661604125247 44.
↪393684955690581,-103.767294323528063 44.393734806102117,-103.767623238680557 44.
↪394127721518785,-103.769273719703676 44.394900867042516,-103.769609703946827 44.
↪395326786724503,-103.769732072038536 44.395745219647871,-103.769609607364416 44.
↪396194309461826,-103.769310708537489 44.396691166475954,-103.768865902286791 44.
↪397236074649896)",1,5,unimproved road,roads
```

At this point the CSV can be imported into SOLR using CURL:

```
curl "http://solr.geo-solutions.it/solr/collection1/update/csv?commit=true&separator=
↪%2C&fieldnames=geo,id,cat_i,label_s,layer_s&header=true"
     -H 'Content-type:text/csv; charset=utf-8' --data-binary @/tmp/roads.csv
```

Some observations:

- The files gets uploaded as a `text/csv` file, older versions might require a `text/plain` mime type
- The `fieldnames` overrides the CSV header and allows us to specify the field name as expected by SOLR

At this point it's possible to configure a layer showing only the roads in the GeoServer UI:

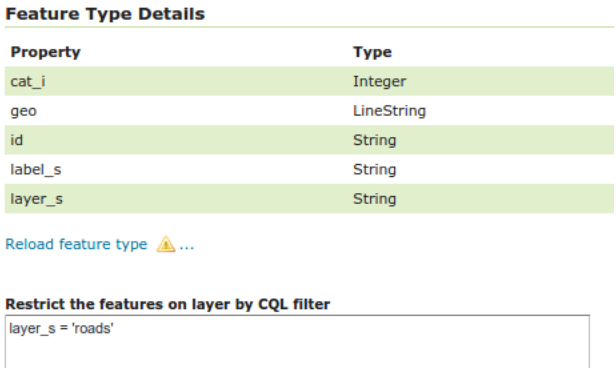


Fig. 16.11: *Setting up the roads layer*

After setting the bounding box and the proper style, the layer preview will show the roads stored in SOLR:

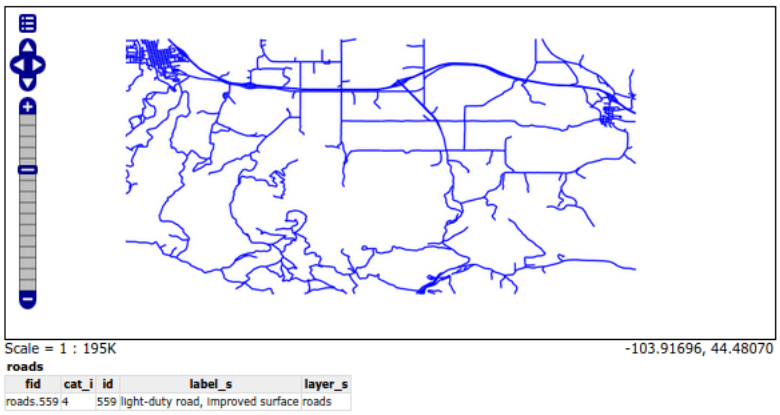


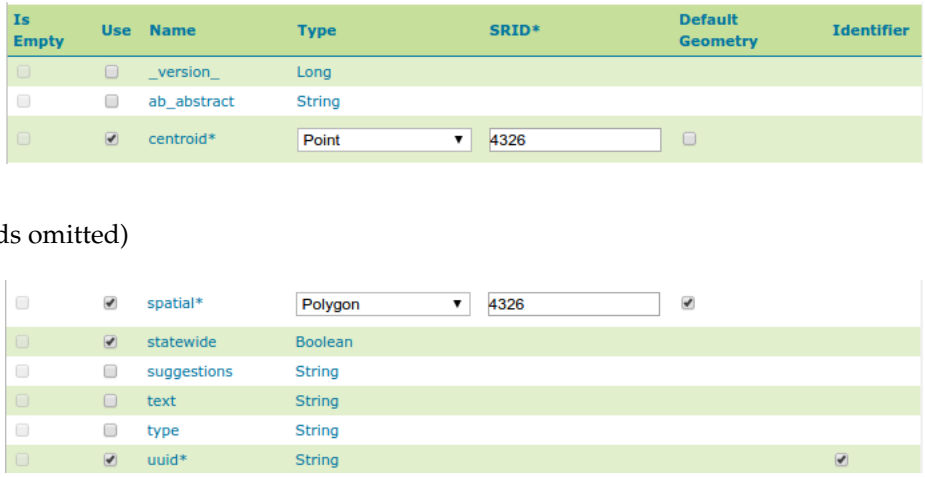
Fig. 16.12: *Preview roads from SOLR layer*

16.14.3 Optimize rendering of complex polygons

Rendering large maps with complex polygons, to show the overall distribution of the data, can take a significant toll, especially if GeoServer cannot connect to the SOLR server via a high speed network.

A common approach to handle this issue is to add a second geometry to the SOLR documents, representing the centroid of the polygon, and using that one to render the features when fairly zoomed out.

Once the SOLR documents have been updated with a centroid column, and it has been populated, the column can be added as a secondary geometry. Make sure to keep the polygonal geometry as the default one:



... (other fields omitted)

Fig. 16.13: *Configuring a layer with multiple geometries*

With this setup the polygonal geometry will still be used for all spatial filters, and for rendering, unless the style otherwise specifical demands for the centroid.

Then, a style with scale dependencies can be setup in order to fetch only then centroids when fairly zoomed out, like in the following CSS example:

```
[@scale > 50000] {
  geometry: [centroid];
  mark: symbol(square);
}
:mark {
  fill: red;
  size: 3;
}
[@scale <= 50000] {
  fill: red;
  stroke: black;
}
```

Using this style the `spatial` field will still be used to resolve the BBOX filter implicit in the WMS requests, but only the much smaller `centroid` one will be transferred to GeoServer for rendering.

16.15 GeoMesa data store

GeoMesa provides a GeoTools DataStore to access SimpleFeatures stored in Apache Accumulo.

16.16 GWC Distributed Caching community module

GWC Distributed Caching module provides support for distributed in Memory caching using the Hazelcast API.

16.16.1 Installing the WPS download module

1. Download the GWC Distributed module from the nightly GeoServer community module builds.

   Warning: Make sure to match the version of the extension to the version of the GeoServer instance.

2. Extract the contents of the archive into the `WEB-INF/lib` directory of the GeoServer installation.

16.16.2 Module description

More information about Caching Configuration can be found at the *Configuration* and *Caching defaults* pages.

16.17 GDAL based WCS Output Format

The gdal_translate based output format leverages the availability of the gdal_translate command to allow the generation of more output formats than GeoServer can natively produce. The basic idea is to dump to the file system a file that gdal_translate can translate, invoke it, zip and return the output of the translation.

This extension is thus the equivalent of the *OGR extension* for raster data.

16.17.1 Out of the box behaviour

Out of the box the plugin assumes the following:

- gdal_translate is available in the path
- the GDAL_DATA variable is pointing to the GDAL data directory (which stores the spatial reference information for GDAL)

In the default configuration the following formats are supported:

- JPEG-2000 part 1 (ISO/IEC 15444-1)
- Geospatial PDF
- Arc/Info ASCII Grid
- ASCII Gridded XYZ

The list might be shorter if gdal_translate has not been built with support for the above formats (for example, the default JPEG-2000 format relies on the JasPer-based GDAL driver).

Once installed in GeoServer, a bunch of new supported formats will be listed in the `ServiceMetadata` section of the WCS 2.0 GetCapabilities document, e.g. `image/jp2` and `application/pdf`.

16.17.2 gdal_translate conversion abilities

The gdal_translate utility is usually able to convert more formats than the default setup of this output format allows for, but the exact list depends on how the utility was built from sources. To get a full list of the formats available by your ogr2ogr build just run:

```
gdal_translate --long-usage
```

and you'll get the full set of options usable by the program, along with the supported formats.

For example, the above produces the following output using gdal 1.11.2 compiled with libgeotiff 1.4.0, libpng 1.6, libjpeg-turbo 1.3.1, libjasper 1.900.1 and libecwj2 3.3:

```
Usage: gdal_translate [--help-general] [--long-usage]
    [-ot {Byte/Int16/UInt16/UInt32/Int32/Float32/Float64/
          CInt16/CInt32/CFloat32/CFloat64}] [-strict]
    [-of format] [-b band] [-mask band] [-expand {gray|rgb|rgba}]
    [-outsize xsize[%] ysize[%]]
    [-unscale] [-scale[_bn] [src_min src_max [dst_min dst_max]]]* [-exponent[_bn] exp_
↪val]*
    [-srcwin xoff yoff xsize ysize] [-projwin ulx uly lrx lry] [-epo] [-eco]
    [-a_srs srs_def] [-a_ullr ulx uly lrx lry] [-a_nodata value]
    [-gcp pixel line easting northing [elevation]]*
    [-mo "META-TAG=VALUE"]* [-q] [-sds]
    [-co "NAME=VALUE"]* [-stats] [-norat]
    src_dataset dst_dataset

GDAL 1.11.2, released 2015/02/10

The following format drivers are configured and support output:
  VRT: Virtual Raster
  GTiff: GeoTIFF
  NITF: National Imagery Transmission Format
  HFA: Erdas Imagine Images (.img)
  ELAS: ELAS
  AAIGrid: Arc/Info ASCII Grid
  DTED: DTED Elevation Raster
  PNG: Portable Network Graphics
  JPEG: JPEG JFIF
  MEM: In Memory Raster
  GIF: Graphics Interchange Format (.gif)
  XPM: X11 PixMap Format
  BMP: MS Windows Device Independent Bitmap
  PCIDSK: PCIDSK Database File
  PCRaster: PCRaster Raster File
  ILWIS: ILWIS Raster Map
  SGI: SGI Image File Format 1.0
  SRTMHGT: SRTMHGT File Format
  Leveller: Leveller heightfield
  Terragen: Terragen heightfield
  ISIS2: USGS Astrogeology ISIS cube (Version 2)
  ERS: ERMapper .ers Labelled
  ECW: ERDAS Compressed Wavelets (SDK 3.x)
  JP2ECW: ERDAS JPEG2000 (SDK 3.x)
  FIT: FIT Image
  JPEG2000: JPEG-2000 part 1 (ISO/IEC 15444-1)
  RMF: Raster Matrix Format
  RST: Idrisi Raster A.1
  INGR: Intergraph Raster
```

```
  GSAG: Golden Software ASCII Grid (.grd)
  GSBG: Golden Software Binary Grid (.grd)
  GS7BG: Golden Software 7 Binary Grid (.grd)
  R: R Object Data Store
  PNM: Portable Pixmap Format (netpbm)
  ENVI: ENVI .hdr Labelled
  EHdr: ESRI .hdr Labelled
  PAux: PCI .aux Labelled
  MFF: Vexcel MFF Raster
  MFF2: Vexcel MFF2 (HKV) Raster
  BT: VTP .bt (Binary Terrain) 1.3 Format
  LAN: Erdas .LAN/.GIS
  IDA: Image Data and Analysis
  LCP: FARSITE v.4 Landscape File (.lcp)
  GTX: NOAA Vertical Datum .GTX
  NTv2: NTv2 Datum Grid Shift
  CTable2: CTable2 Datum Grid Shift
  KRO: KOLOR Raw
  ARG: Azavea Raster Grid format
  USGSDEM: USGS Optional ASCII DEM (and CDED)
  ADRG: ARC Digitized Raster Graphics
  BLX: Magellan topo (.blx)
  Rasterlite: Rasterlite
  PostGISRaster: PostGIS Raster driver
  SAGA: SAGA GIS Binary Grid (.sdat)
  KMLSUPEROVERLAY: Kml Super Overlay
  XYZ: ASCII Gridded XYZ
  HF2: HF2/HFZ heightfield raster
  PDF: Geospatial PDF
  ZMap: ZMap Plus Grid
```

The full list of formats that gdal_translate is able to support is available on the GDAL site. Mind that this output format can handle only outputs that are file based and that do support creation. So, for example, you won't be able to use the PostGIS Raster output (since it's database based) or the Arc/Info Binary Grid (creation not supported).

16.17.3 Customisation

If gdal_translate is not available in the default path, the GDAL_DATA environment variable is not set, or if the output formats needs tweaking, a `gdal_translate.xml` configuration file can be created to customize the output format. The file should be put inside a `gdal` folder in the root of the GeoServer data directory.

Note: GeoServer will automatically detect any change to the file and reload the configuration, without a need to restart.

The default configuration is equivalent to the following xml file:

```
<ToolConfiguration>
  <executable>gdal_translate</executable>
  <environment>
    <variable name="GDAL_DATA" value="/usr/local/share/gdal" />
  </environment>
  <formats>
    <Format>
```

```
      <toolFormat>JPEG2000</toolFormat>
      <geoserverFormat>GDAL-JPEG2000</geoserverFormat>
      <fileExtension>.jp2</fileExtension>
      <singleFile>true</singleFile>
      <mimeType>image/jp2</mimeType>
      <type>binary</type> <!-- not really needed, it's the default -->
      <option>-co</option>
      <option>FORMAT=JP2</option>
    </Format>
    <Format>
      <toolFormat>PDF</toolFormat>
      <geoserverFormat>GDAL-PDF</geoserverFormat>
      <fileExtension>.pdf</fileExtension>
      <singleFile>true</singleFile>
      <mimeType>application/pdf</mimeType>
      <formatAdapters>
             <GrayAlphaToRGBA/>
             <PalettedToRGB/>
      </formatAdapters>
    </Format>
    <Format>
      <toolFormat>AAIGrid</toolFormat>
      <geoserverFormat>GDAL-ArcInfoGrid</geoserverFormat>
      <fileExtension>.asc</fileExtension>
      <singleFile>false</singleFile>
    </Format>
    <Format>
      <toolFormat>XYZ</toolFormat>
      <geoserverFormat>GDAL-XYZ</geoserverFormat>
      <fileExtension>.txt</fileExtension>
      <singleFile>true</singleFile>
      <mimeType>text/plain</mimeType>
      <type>text</type>
    </Format>
  </formats>
</ToolConfiguration>
```

The file showcases all possible usage of the configuration elements:

- `executable` can be just gdal_translate if the command is in the path, otherwise it should be the full path to the executable. For example, on a Linux box with a custom build GDAL library might be:

  ```
  <executable>/usr/local/bin/gdal_translate</executable>
  ```

- `environment` contains a list of `variable` elements, which can be used to define environment variables that should be set prior to invoking gdal_translate. For example, to setup a GDAL_DATA environment variable pointing to the GDAL data directory, the configuration might be:

  ```
  <environment>
   <variable name="GDAL_DATA" value="/usr/local/share/gdal" />
  </environment>
  ```

- `Format` defines a single format, which is defined by the following tags:
  - `toolFormat`: the name of the format to be passed to gdal_translate with the -of option (case insensitive).
  - `geoserverFormat`: is the name of the output format as advertised by GeoServer

- `fileExtension`: is the extension of the file generated after the translation, if any (can be omitted)
- `option`: can be used to add one or more options to the gdal_translate command line. As you can see by the JPEG2000 example, each item must be contained in its own `option` tag. You can get a full list of options by running `gdal_translate --help` or by visiting the GDAL web site). Also, consider that each format supports specific creation options, listed in the description page for each format (for example, here is the JPEG2000 one).
- `singleFile`: if true the output of the conversion is supposed to be a single file that can be streamed directly back without the need to wrap it into a zip file
- `mimeType`: the mime type of the file returned when using `singleFile`. If not specified `application/octet-stream` will be used as a default.
- `formatAdapters`: transformations on the coverage that might need to be applied in order to successfully encode the output. The transformations are applied only if their input conditions are met.

The available format adapters are:

- `GrayAlphaToRGBA`: expands a gray image with alpha channel to RGBA (mandatory for geospatial PDF for example)
- `PalettedToRGB`: expands a paletted image RGB(A) (mandatory for geospatial PDF for example)

16.18 GWC S3 BlobStore plugin

This plugin supports the use of the AWS Simple Storage Service (Amazon S3) as storage medium for *GeoWebCache settings*.

16.18.1 Installing the S3 BlobStore plugin

1. Download the extension from the nightly GeoServer community module builds.

   Warning: Make sure to match the version of the extension to the version of the GeoServer instance!

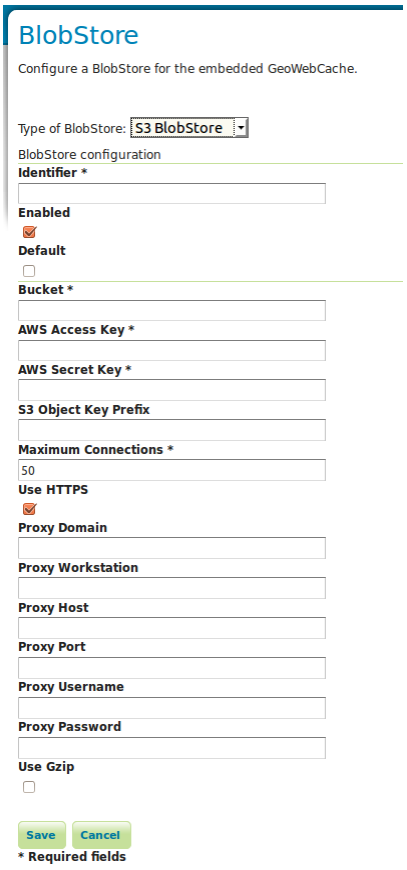
2. Extract the contents of the archive into the `WEB-INF/lib` directory of the GeoServer installation.

16.18.2 Configuring the S3 BlobStore plugin

Once the plugin has been installed, one or more S3 BlobStores may be configured through *BlobStores*. Afterwards, cached layers can be explicitely assigned to it or one blobstore could be marked as 'default' to use it for all unassigned layers.

Bucket

The name of the AWS S3 bucket where the tiles are stored.


BlobStore
Configure a BlobStore for the embedded GeoWebCache.
Type of BlobStore: S3 BlobStore
BlobStore configuration
Identifier *
Enabled
Default
Bucket *
AWS Access Key *
AWS Secret Key *
S3 Object Key Prefix
Maximum Connections *
50
Use HTTPS
Proxy Domain
Proxy Workstation
Proxy Host
Proxy Port
Proxy Username
Proxy Password
Use Gzip
Save
Cancel
* Required fields

AWS Access Key

The AWS Access Key ID.

AWS Secret Key

AWS Secret Access Key.

S3 Object Key Prefix

A prefix path to use as the root to store tiles under the bucket (optional).

Maximum Connections

The maximum number of allowed open HTTP connections.

Use HTTPS

When enabled, a HTTPS connection will be used. When disabled, a regular HTTP connection will be used.

Proxy Domain

A Windows domain name for configuring NTLM proxy support (optional).

Proxy Workstation

A Windows workstation name for configuring NTLM proxy support (optional).

Proxy Host

Proxy host the client will connect through (optional).

Proxy Port

Proxy port the client will connect through (optional).

Proxy Username

User name the client will use if connecting through a proxy (optional).

Proxy Password

Password the client will use if connecting through a proxy (optional).

Use Gzip

When enabled, the stored tiles will be GZIP compressed.

16.19 GWC SQLite Plugin

This plugin provides integration with GWC SQLite based blob stores. At the moment only one blob store of this type is available, the MBTiles blob store.

16.19.1 MBTiles Blob Store

This blob store allow us to store tiles using the MBTiles specification (version 1.1) which defines a schema for storing tiles in an SQLite database with some restrictions regarding tiles formats and projections.

MBTiles specification only supports JPEG and PNG formats and projection EPSG:3857 is assumed. The implemented blob store will read and write MBTiles files compliant with the specification but will also be able to write and read MBTiles files that use others formats and projections.

Using the MBTiles blob store will bring several benefits at the cost of some performance loss. The MBTiles storage uses a significantly smaller number of files, which results in easier data handling (e.g., backups, moving tiles between environments). In some cases the stored data will be more compact reducing the size of the data on disk.

When compared to the file blob store this store has two limitations:

- This store does not integrate with disk quota, this is a consequence of using database files.
- **This store cannot be shared among several GeoWebCache instances.**

Note: If disk quota is activated the stored stats will not make much sense and will not reflect the actual disk usage, the size of the database files cannot be really controlled.

Database files cannot be managed as simple files. When connections to a database are open the associated file should not be deleted, moved or switched or the database file may become corrupted. Databases files can also become fragmented after deleting an huge amount of data or after frequent inserts, updates or delete operations.

File Path Templates

An MBTiles file will correspond to an SQLite database file. In order to limit the amount of contention on each single database file users will be allowed to decide the granularity of the databases files. When GeoWebCache needs to map a tile to a database file it will only look at the databases files paths, it will not take in account the MBTiles metadata (this is why this store is able to handle others formats and projections).

To configure the databases files granularity the user needs to provide a file path template. The default file path template for the MBTiles blob store is this one:

```
{layer}/{grid}{format}{params}/{z}-{x}-{y}.sqlite
```

This file template will stores all the tiles belonging to a certain layer in a single folder that will contain sub folders for each given format, projection and set of parameters and will group tiles with the same zoom level, column range and row range in a SQLite file. The column and row range values are passed by configuration, by default those values are equal to 250. The provided files paths templates will always be considered relative to the root directory provided as a configuration option.

Follows an example of what the blob store root directory structure may look like when using the default path template:

```
.
|-- nurc_Pk50095
|   `-- EPSG_4326image_pngnull
|       |-- 11_2000_1500.sqlite
|       `-- 12_4250_3000.sqlite
`-- topp_states
    |-- EPSG_900913image_jpeg7510004a12f49fdd49a2ba366e9c4594be7e4358
    |   |-- 6_250_500.sqlite
    |   `-- 7_0_0.sqlite
    `-- EPSG_900913image_jpegnull
        |-- 3_500_0.sqlite
        |-- 4_0_250.sqlite
        `-- 8_750_500.sqlite
```

If no parameters were provided *null* string will be used. Is the responsibility of the user to define a file path template that will avoid collisions.

The terms that can be used in the file path template are:

- **grid**: the grid set id
- **layer**: the name of the layer
- **format**: the image format of the tiles
- **params**: parameters unique hash
- **x**: column range, computed based on the column range count configuration property
- **y**: row range, computed based on the row range count configuration property
- **z**: the zoom level

It is also possible to use parameters values, like *style* for example. If the parameter is not present *null* will be used.

Note: Characters `\` and `/` can be used as path separator, they will be translated to the operating system specific one (`\` for Linux and `/` for Windows). Any special char like `\`, `/`, `:` or empty space used in a term value will be substituted with an underscore.

MBTiles Metadata

A valid MBTiles file will need some metadata, the image format and layer name will be automatically added when an MBTiles file is created. The user can provide the remaining metadata using a properties file whose name must follow this pattern:

```
<layerName>.metadata
```

As an example, to add metadata `description` and `attribution` entries to layer `tiger_roads` a file named `tiger_roads.properties` with the following content should be present in the metadata directory:

```
description=ny_roads
attribution=geoserver
```

The directory that contains this metadata files is defined by a configuration property.

Expiration Rules

The MBTiles specification don't give information about when a tile was created. To allow expire rules, an auxiliary table is used to store tile creation time. In the presence of an MBTiles file generated by a third party tool it is assumed that the creation time of a tile was the first time it was accessed. This feature can be activated or deactivated by configuration. Note that this will not break the MBTiles specification compliance.

Eager Truncate

When performing a truncate of the cache the store will try to remove the whole database file avoiding to create fragmented space. This is not suitable for all the situations and is highly dependent on the database files granularity. The configuration property `eagerDelete` allows the user to disable or deactivate this feature which is disabled by default.

When a truncate request by tile range is received all the the databases files that contains tiles that belong to the tile range are identified. If eager delete is set to true those databases files are deleted otherwise a single delete query for each file is performed.

Configuration Example

Follows as an example the default configuration of the MBTiles store:

The *rootDirectory* property defines the location where all the files produced by this store will be created. The *templatePath* property is used to control the granularity of the database files (see section above). Properties *rowRangeCount* and *columnRangeCount* will be used by the path template to compute tile ranges.

The *poolSize* property allows to control the max number of open database files, when defining this property the user should take in account the number open files allowed by the operating system. The *poolReaperIntervalMs* property controls how often the pool size will be checked to see if some database files connections need to be closed.

Property *eagerDelete* controls how the truncate operation is performed (see section above). The property *useCreateTime* can be used to activate or deactivate the insertion of the tile creation time (see section above). Property *executorConcurrency* controls the parallelism used to perform certain operations, like the truncate operation for example. Property *mbtilesMetadataDirectory* defines the directory where the store will look for user provided MBTiles metadata.

Note: Since the connection pool eviction happens at a certain interval, it means that the number of files open concurrently can go above the threshold limit for a certain amount of time.

Replace Operation

As said before, if the cache is running SQLite files cannot be simply switched, first all connections need to be closed. The replace operation was created for this propose. The replace operation will first copy the new file side by side the old one, then block the requests to the old file, close the connections tot he old file, delete the old one, rename the new file to current one, reopen the new db file and start serving requests again. Should be almost instant.

A REST entry point for this operation is available, it will be possible to submit a ZIP file or a single file along with the request. The replace operation can also use an already present file or directory. When using a directory the directory structure will be used to find the destination of each file, all the paths will be assumed to be relative to the store root directory. This means that is possible to replace a store content with


BlobStore
Configure a BlobStore for the embedded GeoWebCache.
Type of BlobStore:
MBTiles BlobStore
BlobStore configuration
Identifier *
mbtiles-store
Enabled
Default
Root Directory *
/tmp/gwc-tests-a
Browse...
Path Template
{grid}/{layer}/{format}/{params}/{z}/tiles_{x}_
Row Range Count
250
Column Range Count
250
Pool Size
1000
Pool Reaper Interval in Milliseconds
500
Eager Delete
Use Create Time
MBTiles Metadata Directory
/tmp/gwc-mbtiles/layersMetadata
Browse...
Internal Executor Concurrency
5
Save
Cancel
* Required fields

another store content (a seeded one for example) by zipping the second store root directory and send it as a replacement.

Note: When using a local directory or submitting a zip file all the file present in the directory will be considered.

There is four ways to invoke this operation. Follows an example of all those variants invocations using CURL.

Replace a single file uploading the replacement file:

```
curl -u admin:geoserver -H 'Content-Type: multipart/form-data'
  -F "file=@tiles_0_0.sqlite"
  -F "destination=EPSG_4326/sf_restricted/image_png/null/10/tiles_0_0.sqlite"
  -F "layer=sf:restricted"
  -XPOST 'http://localhost:8080/geoserver/gwc/rest/sqlite/replace'
```

Replace a single file using a file already present on the system:

```
curl -u admin:geoserver -H 'Content-Type: multipart/form-data'
  -F "source=/tmp/tiles_0_0.sqlite"
  -F "destination=EPSG_4326/sf_restricted/image_png/null/10/tiles_0_0.sqlite"
  -F "layer=sf:restricted"
  -XPOST 'http://localhost:8080/geoserver/gwc/rest/sqlite/replace'
```

Replace multiple files uploading a ZIP file:

```
curl -u admin:geoserver -H 'Content-Type: multipart/form-data'
  -F "file=@tiles.zip"
  -F "layer=sf:restricted"
  -XPOST 'http://localhost:8080/geoserver/gwc/rest/sqlite/replace'
```

Replace multiple files using a directory already present on the system:

```
curl -u admin:geoserver -H 'Content-Type: multipart/form-data'
  -F "source=/tmp/tiles"
  -F "layer=sf:restricted"
  -XPOST 'http://localhost:8080/geoserver/gwc/rest/sqlite/replace'
```

The *layer* parameter identifies the layer whose associated blob store content should be replaced. The *file* parameter is used to upload a single file or a ZIP file. The *source* parameter is used to reference an already present file or directory. The *destination* parameter is used to define the file that should be replaced with the provided file.

This are the only valid combinations of this parameters other combinations will ignore some of the provided parameters or will throw an exception.

16.20 Parameters Extractor

This module allow us to entering specific request parameters as URL path fragments instead of using the query string. For example, we want to be able to apply a cql_filter using a URL in the following form:

```
/geoserver/<workspace>/<layer>/<filter>/ows?service=WMS&version=1.3.0&request=GetMap
```

As a simple example of usage, if the <filter> is something like:

```
K_140M
```

the URL would become:

```
/geoserver/<workspace>/<layer>/K_140M/ows?service=WMS&version=1.3.0&request=GetMap
```

and this module will translate the URL to this new one:

```
/geoserver/<workspace>/<layer>/ows?service=WMS&version=1.3.0&request=GetMap&cql_
↪filter=seq='K140M'
```

This module is configured by a set of rules that will be applied to the incoming URLs. Note that a get capabilities result will include the original URL maintaining the extra filter.

This module also gives the possibility to echo existing URL parameters to the result of a get capabilities result. As an example, by default the following get capabilities request (note the existing cql_filter parameter):

```
/geoserver/ows?service=wms&version=1.3.0&request=GetCapabilities&cql_filter=CFCC=
↪%27D68%27
```

will return a get capabilities document were the URLs will be of the type:

```
/geoserver/ows?SERVICE=WMS&
```

if this module is configured to echo an existing cql_filter parameter the result would be:

```
/geoserver/ows?SERVICE=WMS&CQL_FILTER=CFCC%3D%27D68%27&
```

This module is configured using three types of rules: echo parameter rules, basic rules and advanced rules. All of them can be managed in this module UI which is integrated in GeoServer UI.

16.20.1 Echo Parameter Rules

Echo parameter rules are very simple, they allow us to define that a certain existing URL parameter should be echoed to a get capabilities result. This type of rules only required one mandatory parameter which is the name of the existing URL parameter that should be echoed to a get capabilities result.

Example of an echo parameter rule:

Fig. 16.14: *Example of a echo parameter rule defined in the UI*

This rule will echo the cql_filter of this URL:

```
/geoserver/ows?service=wms&version=1.3.0&request=GetCapabilities&cql_filter=CFCC=
↪%27D68%27
```

to the get capabilities result:

```
/geoserver/ows?SERVICE=WMS&CQL_FILTER=CFCC%3D%27D68%27&
```

16.20.2 Basic Rules

Basic rules allow us to handle simple uses cases where we only want to extract a parameter from the URL.

A basic rule is defined by three mandatory attributes:

Attribute	Description
`Position`	The position of the URL base path element to be selected
`Parameter`	The name of the parameter produced by this rule
`Transform`	Expression that defines the value of the parameter, use {PARAMETER} as a placeholder for the selected path element

For commodity is also possible when defining this type of rules to configure that an existing parameter in the URL should be echoed to a get capabilities result.

Example of a basic rule:

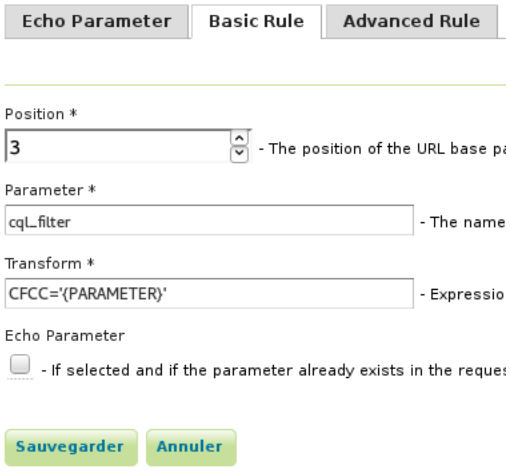


Fig. 16.15: *Example of a basic rule defined in the UI*

This rule will transform the URL:

```
/geoserver/tiger/wms/H11?SERVICE=WMS&VERSION=1.1.1&REQUEST=GetMap
```

in:

```
/geoserver/tiger/wms?SERVICE=WMS&VERSION=1.1.1&REQUEST=GetMap&CQL_FILTER=CFCC%3D%27H11
↪%27
```

16.20.3 Advanced Rules

Advanced rules allow us to handle more complex uses cases where more flexibility is required.

An advanced rule is defined by three mandatory attributes and four optional ones:

Attribute	Description	Mandatory
Match	Regex match expression with groups, for example ^(?:/[^/]*){3}(/([^/]+)).*$ selects the URL base path third element	Yes
Activation	If defined this rule will only be applied to URLs that match this regex expression	No
Parameter	The name of the parameter produced by this rule	Yes
Transform	Expression that defines the value of the parameter, use $1 ... $n as placeholders for groups defined in the match expression	Yes
Remove	The match expression group to be removed from URL, by default 1	No
Combine	Defines how to combine parameter existing value ($1 existing value, $2 new value), by default the value is overridden	No

For commodity is also possible when defining this type of rules to configure that an existing parameter in the URL should be echoed to a get capabilities result.

Example of an advanced rule:

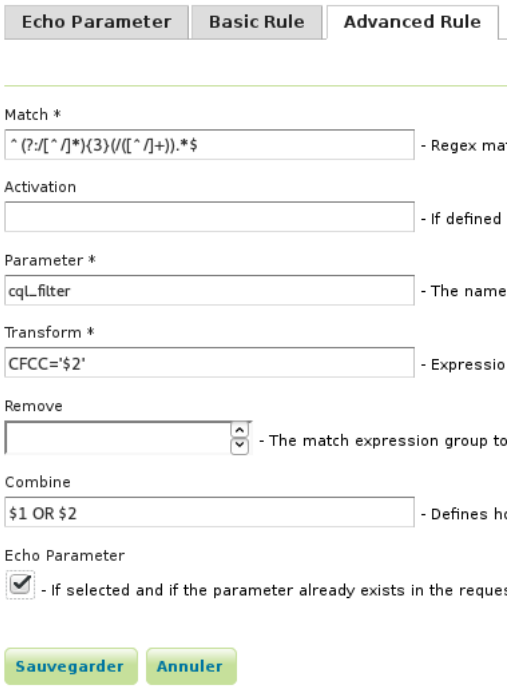


Fig. 16.16: *Example of an advanced rule defined in the UI*

This rule will transform the URL:

```
/geoserver/tiger/wms/H11?SERVICE=WMS&VERSION=1.1.1&REQUEST=GetMap&CQL_FILTER=CFCC%3D
↪%27D68%27
```

in:

```
/geoserver/tiger/wms?SERVICE=WMS&VERSION=1.1.1&REQUEST=GetMap&CQL_FILTER=CFCC%3D%27D68
↪%27+or+CFCC%3D%27H11%27
```

No that this rule will also echo an existing cql_filter parameter to the get capabilities result.

16.20.4 Rules Management

Rules can be managed and tested in the rules management UI. Besides the basic operations like add, remove and update is also possible to activate or deactivate rules. A deactivated rule will be ignored by this module.

Follow a print screen of the rules management UI with all the rules previously defined:

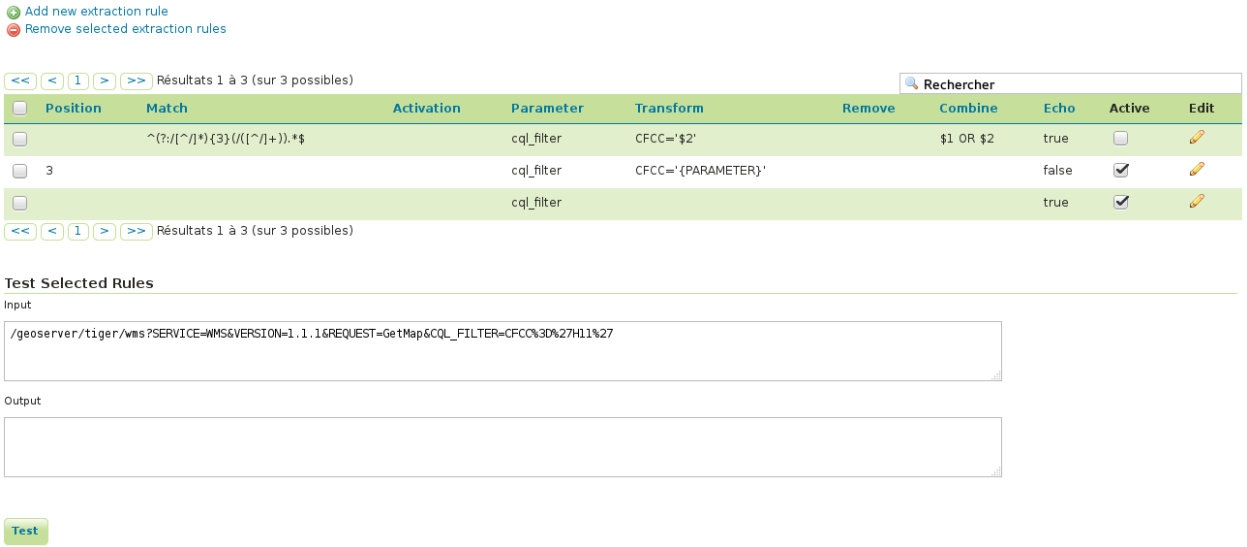


Fig. 16.17: *Rules management UI*

Note that the first rule (the advanced one) is not active.

16.20.5 REST API

The rules and echo parameters can also be managed by means of a REST API found at `geoserver/rest/params-extractor`. Documentation for it is available in Swagger format

16.21 WPS Remote community module

The GeoServer WPS remote module allows to discover, run and monitor processes running on one or more remote machines, exposing them via the WPS protocol and allowing both synchronous and asynchronous runs of the same, with eventual progress monitoring.

The remote process can be anything, from a Python script or a command line executable. The only constraint is to have a remote component able to handle few RPCs, like run, progress, complete (which means collect and send the outcome to the GeoServer machine), execution error (which means if any error occurs report the exception) and kill. On GeoServer side the module manages the same RPCs in order to perform the integration with the WPS. All the communications and command take place over the XMPP protocol, as a suitable cross-language communication system

A reference implementation of the remote end is available at https://github.com/geoserver/wps-remote, a configurable Python/XMPP wrapper for remote commands. The Python XMPP wrapper resides into the remote machine and is able to send a presentation of the remote process through an XMPP message by JSON-encoding into the body the process inputs/outputs parameter descriptors along with their type. On the GeoServer side the WPS Remote module automatically recognizes and loads an XMPP implementation

of the RemoteClient. The GeoServer plugin is able to inquire for new available services, un-marshall their inputs and outputs and build appropriate process wrapper for GeoServer WPS to use. At execution time, the new Process is able to interact with the RemoteClient plug-in implementation in order to send a request to the Service.py, follow the status of the remote process and get the outputs at the end.

16.21.1 Orchestrating remote executable through GeoServer WPS and XMPP

The Remote WPS infrastructure is designed to run external programs **asynchronously** through the standard *OGC WPS* protocol interface on remote machines and to track their progress through the *XMPP Protocols*.

The infrastructure relies on an *XMPP Server* (which can be external or embedded) in order to implement a *message passing distributed infrastructure* that uses the XMPP protocols for exchanging control and status messages between the GeoServer instances and the executables running on remote machines.

A Python based framework, along with addoc GeoServer plugins, manages and translates the XMPP messages by decoupling the custom mechanism of launching external executables running on remote machines from the standard way of invoking OGC WPS compliant processes in GeoServer.

The OGC WPS process I/O map is automatically generated by the Remote WPS GeoServer plug-ins at runtime. Every time the framework recognizes a new external executable declared on the XMPP channel, the Remote WPS GeoServer plug-in creates the equivalent OGC WPS compliant interface and establishes the communication between GeoServer and the external executable. Every time a GeoServer user starts a new OGC WPS compliant process run, the infrastructure performs a call to the external executable and takes care of managing the communication between the two entities asynchronously.

The Remote WPS GeoServer plug-ins provide a new *ProcessFactory* which is responsible to register/unregister the OGC WPS compliant processes automatically and at runtime. Orchestration will be performed by the newly created GeoServer ProcessFactory upon a WPS execution request with the help of the XMPP Server which provides out-of-the-box nodes presence discovery and load balancing capabilities for the remote node through the interaction with the Python framework.

The orchestrator will also be responsible for redirecting the messages generated by running executables on the remote machines to the correct GeoServer process and vice versa for control messages. Executables running on the remote hosts through the Python framework wrappers, will generate progress information which will be sent back to the orchestrator via XMPP.

It is important to note that there are two levels of load balancing. One on the GeoServer side and one on the remote processing nodes side. The GeoServer load balancer will know which user has requested the processing and will be able to enforce system and resources limitations associated to it; as an instance, a user won't be able to run more than M processes in parallel and a quota for both the inputs and outputs will be assigned to its executions. The Orchestrator will be responsible, in cooperation with the XMPP service, for balancing the load for a certain executables on the remote hosts and for deciding where to send a new execution request in case a certain executable has been deployed on multiple remote machines.

Also note that the GeoServer instances comprising the cluster will share a specific directory where the WPS specific resources will be stored.

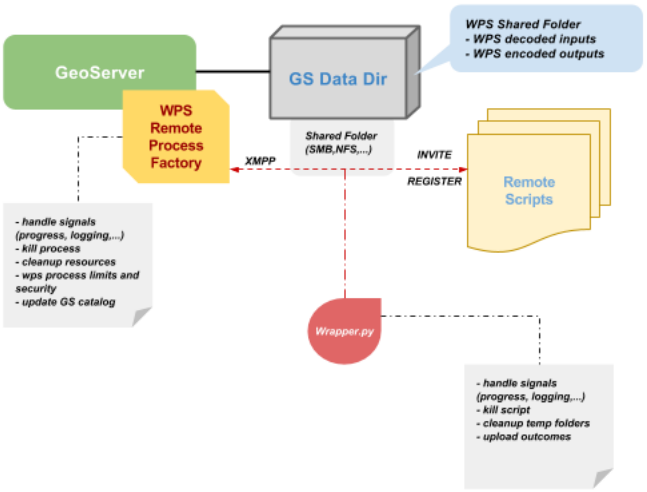
Invoking remote processes and consuming the results

The executables on the remote hosts will be invoked through the command line by the Python framework wrappers in response to execution commands, received via the Orchestrator XMPP messages, as the result of a GeoServer WPS Execute call.

On the remote machines the Python framework wrappers and scripts will act as XMPP clients providing the ability to:

- launch command-line executables using the inputs provided by the end user from the GeoServer process orchestrator
- send back to the orchestrator status messages with the progress and status of the execution
- receive eventual control messages from the orchestrator to tentatively kill running executions
- perform clean-up operations like looking for zombie executions and killing them or removing stale files on the file system from old executions
- perform runtime discovery of new remote executable

The illustration below shows the interaction between the GeoServer Remote WPS plug-in and the Python framework wrappers. The whole communication is achieved through the XMPP protocols. The Python framework makes available a set of scripts and wrappers allowing to invoke the external executables and manage the entire execution by translating commands and outputs into XMPP messages to be sent and received by GeoServer.



As mentioned above the remote executables should adhere to a certain contract in order to fully support all these functionalities. As an example the ability to kill an existing execution heavily relies on the fact that the current process accounted for this functionality, otherwise we would have to try and kill it using operating system calls. This might require the end user to create wrappers around the legacy executables in certain cases.

Runtime discovery of new remote executable will be supported through adding new Properties files to a location known to the Python modules. A new OGC WPS compliant process will be directly exposed to GeoServer without restarting the GeoServer itself for the new remote executable since the Python wrappers will communicate the existence of a new executable by interacting with the GeoServer Orchestrator via XMPP messages. The end user will see as many OGC WPS compliant processes as properties configuration files on the remote machines thanks to the functionalities implemented by the Python framework scripts.

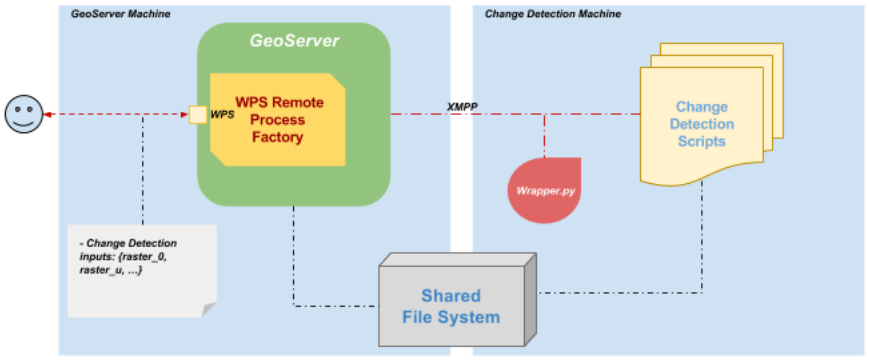
An example of such a properties file can be like below:

```
[Description]
service = Service
namespace = default
description = A dummy service, replace with your own description
executable = process.py
process_buffer = 0
result_size = 0
```

```
active = True

[Options]
customargs = --path=D:\user\
argformat = --key=value
debug = True

[Input]
name = {"type": "string", "description": "A person name", "enum": ["Hans",
↪"Peter", "Alex", "Michi"],
"default": "Hans", "max": 1}
surname = {"type": "string", "description": "A persons surname", "max": 1,
↪"default": "Meier"}
child = {"type": "string", "description": "A child name", "min": 0, "max":␣
↪10}

[Output]
welcome = {"type": "string", "description": "A welcome message"}
goodbye = {"type": "string", "description": "A goodbye message"}
```

Warning: the configuration above is a simplified and non-working example of a Remote Process wrapper configuration. The scope of the example above is just for better understand of the high-level scenario and is not ment to be used in a real use case.

Deploy Diagram



The illustration shows a deploy diagram of a *Change Detection* executable running on a remote machine and exposed as a GeoServer WPS Process thought the *WPS Remote Plug-in*.

- The external users issue a processing request through GeoServer using the OGC WPS compliant protocol.
- The GeoServer Remote WPS plug-in, thanks to the WPS RemoteProcessFactory, will be able to expose the processes I/O map along with the process descriptors, and take care of the entire execution by providing feedbacks to the users in an OGC compliant way.
- On the remote machines, where the executables rely and where the real computation takes place, the Python framework, through the use of wrappers and ad h.o.c. scripts, handles transparently the communication with the Remote plug-in through the XMPP protocols.

- The XMPP Server, in the middle, handles the secured communication channels ensuring that the endpoints are correctly registered to the system and are able to exchange messages.
- The outcomes are exchanged through a shared folder.

16.21.2 Installation and Configuration Steps

The following sections will guide the user to the deployment and configuration of an example *GDAL CONTOUR* command exposed as a GeoServer WPS Process through the WPS Remote Plugin.

The examples will show step-by-step procedures to configure and deploy the whole example on two different machines:

- Commands to deploy GeoServer with the WPS Remote Plug-in and an OpenFire XMPP Server, will be executed on a CentOS Minimal machine
- Commands to deploy the WPS Remote XMPP Python Wrapper and GDAL, will be executed on a Windows 7+ machine

Deployment And Setup Of GeoServer With WPS Remote Plugin

The following commands will prepare a CentOS 7 Minimal ISO machine for the deployment of:

- GeoServer with the following plugins:
  - GeoServer WPS
  - GeoServer Remote WPS Orchestrator
  - GeoServer Importer

The OS iso has been downloaded from::

```
http://isoredirect.centos.org/centos/7/isos/x86_64/CentOS-7-x86_64-Minimal-1503-01.iso
```

Preparation of the system: standard and basic OS packages

Hostname and other useful packages

Update the file `/etc/hosts` making sure that the ip addreses matches the name of the machine.

```
# as root

$> yum -y install man vim openssh-clients mc zip unzip wget net-tools
```

Configure the Java Virtual Environment

```
# as root

$> wget --no-check-certificate --no-cookies --header "Cookie: oraclelicense=accept-
↪securebackupcookie" http://download.oracle.com/otn-pub/java/jdk/8u65-b17/jdk-8u74-
↪linux-x64.tar.gz

$> tar xzvf jdk-8u65-linux-x64.tar.gz
```

```
$> mkdir /usr/java
$> mv jdk1.8.0_65/ /usr/java/

$> alternatives --install /usr/bin/java java /usr/java/jdk1.8.0_65/bin/java 20000

$> alternatives --install /usr/bin/javac javac /usr/java/jdk1.8.0_65/bin/javac 20000

$> alternatives --install /usr/bin/jar jar /usr/java/jdk1.8.0_65/bin/jar 20000

$> alternatives --install /usr/bin/javaws javaws /usr/java/jdk1.8.0_65/bin/javaws␣
↪20000

$> alternatives --set java /usr/java/jdk1.8.0_65/bin/java
$> alternatives --set javac /usr/java/jdk1.8.0_65/bin/javac
$> alternatives --set jar /usr/java/jdk1.8.0_65/bin/jar
$> alternatives --set javaws /usr/java/jdk1.8.0_65/bin/javaws

# Verify the proper installation on the JDK

$> java -version
  java version "1.8.0_65"
  Java(TM) SE Runtime Environment (build 1.8.0_65-b17)
  Java HotSpot(TM) 64-Bit Server VM (build 25.65-b01, mixed mode)

$> javac -version
  javac 1.8.0_65
```

Installing Apache Tomcat

```
# as root

$> yum -y install tomcat-webapps
$> systemctl disable tomcat.service

$> cp /etc/sysconfig/tomcat /etc/sysconfig/geoserver
$> ln -s /usr/share/tomcat/ /opt/tomcat
```

Creating apache tomcat HOME context

Creating base template directory

```
# as root

$> mkdir -p /var/lib/tomcat/geoserver/{bin,conf,logs,temp,webapps,work}

$> cp -Rf /opt/tomcat/conf/* /var/lib/tomcat/geoserver/conf/
```

Creating geoserver apache tomcat BASE context

Make sure you already:

- installed tomcat (Installing apache tomcat)
- created the base catalina template (Creating apache tomcat HOME context)

Edit `server.xml` file

GeoServer is the first tomcat instance we are installing in this VM, so we can keep the default ports:

- 8005 for commands to catalina instance
- 8009 for the AJP connection port
- 8080 for the HTTP connection port

Remember that you may change these ports in the file `/var/lib/tomcat/geoserver/conf/server.xml`

Final configurations

Set the ownership of the `geoserver/` related directories to user tomcat

```
# as root

$> chown tomcat: -R /var/lib/tomcat/geoserver
$> cp /etc/tomcat/tomcat.conf /etc/tomcat/geoserver.conf

$> vi /etc/tomcat/geoserver.conf

  # This variable is used to figure out if config is loaded or not.
  TOMCAT_CFG_LOADED="1"

  # In new-style instances, if CATALINA_BASE isn't specified, it will
  # be constructed by joining TOMCATS_BASE and NAME.
  TOMCATS_BASE="/var/lib/tomcats/"

  # Where your java installation lives
  #JAVA_HOME="/usr/lib/jvm/jre"
  JAVA_HOME="/usr/java/jdk1.7.0_71"

  # Where your tomcat installation lives
  CATALINA_HOME="/usr/share/tomcat"
  CATALINA_BASE="/var/lib/tomcat/geoserver"
  CATALINA_PID=$CATALINA_BASE/work/pidfile.pid

  # System-wide tmp
  CATALINA_TMPDIR="/var/cache/tomcat/temp"

  # You can pass some parameters to java here if you wish to
  #JAVA_OPTS="-Xminf0.1 -Xmaxf0.3"
  # Use JAVA_OPTS to set java.library.path for libtcnative.so
  #JAVA_OPTS="-Djava.library.path=/usr/lib"
  JAVA_OPTS="-server -XX:SoftRefLRUPolicyMSPerMB=36000 -Xms1024m -Xmx2048m
  -XX:PermSize=64m -XX:+UseConcMarkSweepGC -XX:NewSize=48m -DGEOSERVER_DATA_DIR=/
↪storage/data/
  -DENABLE_ADVANCED_PROJECTION=false -Dorg.geotools.shapefile.datetime=true -Duser.
↪timezone=GMT
  -Dorg.geotools.filter.function.simplify=true -DGEOMETRY_COLLECT_MAX_
↪COORDINATES=50000"

  # You can change your tomcat locale here
  #LANG="en_US"
  # Run tomcat under the Java Security Manager
  SECURITY_MANAGER="false"

$> cp /usr/lib/systemd/system/tomcat.service /usr/lib/systemd/system/geoserver.service

$> vi /usr/lib/systemd/system/geoserver.service
```

```
  EnvironmentFile=/etc/tomcat/geoserver.conf

$> systemctl enable geoserver.service
$> systemctl restart geoserver.service

# Follow the server startup procedure and make sure everything goes smoothly through␣
↪the following command

$> tail -F /var/lib/tomcat/geoserver/logs/catalina.YYYY-MM-DD.log
```

Deploy And Configure GeoServer

First deployment

```
# as root

# Git and Maven must be installed on the system
$> yum -y install git
$> yum -y install maven

# Verify the Maven installation and double check that the JDK recognized is the Java␣
↪Sun 1.7+
$> mvn -version

  Apache Maven 3.0.5 (Red Hat 3.0.5-16)
  Maven home: /usr/share/maven
  Java version: 1.8.0_65, vendor: Oracle Corporation
  Java home: /usr/java/jdk1.8.0_65/jre
  Default locale: en_US, platform encoding: UTF-8
  OS name: "linux", version: "3.10.0-229.el7.x86_64", arch: "amd64", family: "unix"

# The following procedures allow to collect and compile the source code from the GIT␣
↪repository.
$> cd

$> git clone https://github.com/geosolutions-it/geoserver.git geoserver.src

$> cd geoserver.src/src

$> git checkout wps-remote
$> git pull

$> mvn clean install -Pwps,wps-remote,importer,security,rest-ext -DskipTests

$> mv web/app/target/geoserver.war /var/lib/tomcat/geoserver/webapps/

$> chown -Rf tomcat: /var/lib/tomcat/geoserver

$> mv /var/lib/tomcat/geoserver/webapps/geoserver/data/ /storage/

$> chown -Rf tomcat: /storage

$> vim /storage/data/remoteProcess.properties

      # Default Properties
      remoteProcessStubCycleSleepTime = 100
```

```
      # Base path where uploaded files are stored
      # . This is used only when a remote uploader is enabled on the Python
      # . WPS Agent. This property represents the local base path (on the filesystem
      # . of GeoServer) where to search for uploaded files.
      # . If not file has been found here (or this option is not enabled), GeoServer
      # . looks for absolute path and/or paths relative to the GEOSERVER DATA DIR.
      #uploadedFilesBasePath = /tmp

      # Full path to the template used to generate the OWS WMC Json output
      # . This property is used only when a "application/owc" output type on
      # . the Python WPS Agent.
      #owc_wms_json_template = absolute_path/to/wmc_template.json

      # Specific kvps for {@link RemoteProcessClient) implementations
      xmpp_server = localhost
      xmpp_server_embedded = false
      xmpp_server_embedded_secure = true
      xmpp_server_embedded_certificate_file = bogus_mina_tls.cert
      xmpp_server_embedded_certificate_password = boguspw
      xmpp_port = 5222

      xmpp_manager_username = admin
      xmpp_manager_password = R3m0T3wP5

      # domain and MUC service name of the XMPP Server
      xmpp_domain = geoserver.org
      xmpp_bus = conference

      # name, user and password of the management room
      xmpp_management_channel = management
      xmpp_management_channel_user = admin
      xmpp_management_channel_pwd = R3m0T3wP5

      # comma separated list of available rooms for services. Those rooms'names will␣
↪be equal to the service and WPS Process namespace
      # Avoid spaces
      xmpp_service_channels = default,geosolutions

      # millis
      xmpp_packet_reply_timeout = 500

      # connection keep alive
      xmpp_connection_ping_interval = 30000
      xmpp_connection_ping_timeout = 10000
      xmpp_connection_ping_initial_delay = 20000

      # Thresholds indicating overloaded resources
      xmpp_cpu_perc_threshold = 82.5
      xmpp_mem_perc_threshold = 84.6

# Restart GeoServer

$> service geoserver restart
```

Warning: GeoServer won't connect to XMPP Server until it has been correctly configured and started as explained in the next section *Installation Of OpenFire XMPP Server To Exchange Messages*.

Installation Of OpenFire XMPP Server To Exchange Messages

The following commands will prepare a CentOS 7 Minimal ISO machine for the deployment of:

- Openfire XMPP Server
- NFS shared file-system

Note: Prerequisite to this section, is the basic preparation of the CentOS machine as described on the section *Deployment And Setup Of GeoServer With WPS Remote Plugin*.

Setup and configuration of Openfire XMPP Server

Originally named Jabber, XMPP is the new label for *Extensible Messaging and Presence Protocol.* and it is associated mostly with instant messaging.

Setting up PostgreSQL database backend

For the purposes of running a private XMPP communication platform, we can safely stick with PostgreSQL 9.2 which is stable and comes in CentOS 7 by default.

```
# as root

$> yum install -y postgresql postgresql-server postgresql-devel postgresql-libs

# After PostgreSQL packages are installed, enable PostgreSQL to start after each␣
↪reboot.

$> systemctl enable postgresql.service

# Initialize directory structure and postgres system database.

$> postgresql-setup initdb

# And start the service.

$> systemctl start postgresql.service
```

Postgres installation is now up and running, lets proceed with setting up the specific database and the dedicated user for OpenFire, together with authentication method and administration password.

For full administration access, switch to postgres user.

```
su postgres

# as postgres

$> createdb openfire
$> createuser -P openfire
```

```
# The '-P' parameter ensures that the shell will explicitly ask for user's password␣
↪and you will need to type it in. Enter the password twice

   R3m0T3wP5

$> psql -U postgres -d postgres -c "ALTER USER postgres WITH PASSWORD 'R3m0T3wP5';"
```

Postgres user is secured with the new password. Lets put authentication methods in practice and force every application or shell login to prompt for these passwords.

```
# as postgres

$> vim /var/lib/pgsql/data/pg_hba.conf

# Scroll down to the bottom of the file and replace all peer and ident strings with␣
↪md5 string.
# The configuration should look like this:

   # TYPE DATABASE USER CIDR-ADDRESS METHOD

   # "local" is for Unix domain socket connections only

   local      all         all                 md5

   # IPv4 local connections:

   host       all         all 127.0.0.1/32    md5

   # IPv6 local connections:

   host       all         all      ::1/128    md5
```

Go back from postgres shell (Ctrl+D) and restart postgresql service as root.

```
# as root

$> systemctl restart postgresql.service
```

Download and install Openfire from Ignite Realtime

Since OpenFire RPM package is not included in any major RHEL / CentOS / Fedora distribution repositories, it must be downloaded directly from Ignite Realtime website.

```
# as root

$> wget http://www.igniterealtime.org/downloadServlet?filename=openfire/openfire-3.10.
↪0-1.i386.rpm -O openfire-3.10.0-1.i386.rpm

# This package come in 32bit version only, so in case we run this installation on x86_
↪64 system, we need to make sure to install coresponding 32bit libraries as well.

$> yum install -y /root/openfire-3.9.3-1.i386.rpm

$> yum install -y glibc.i686
```

Enable the openfire service and start it

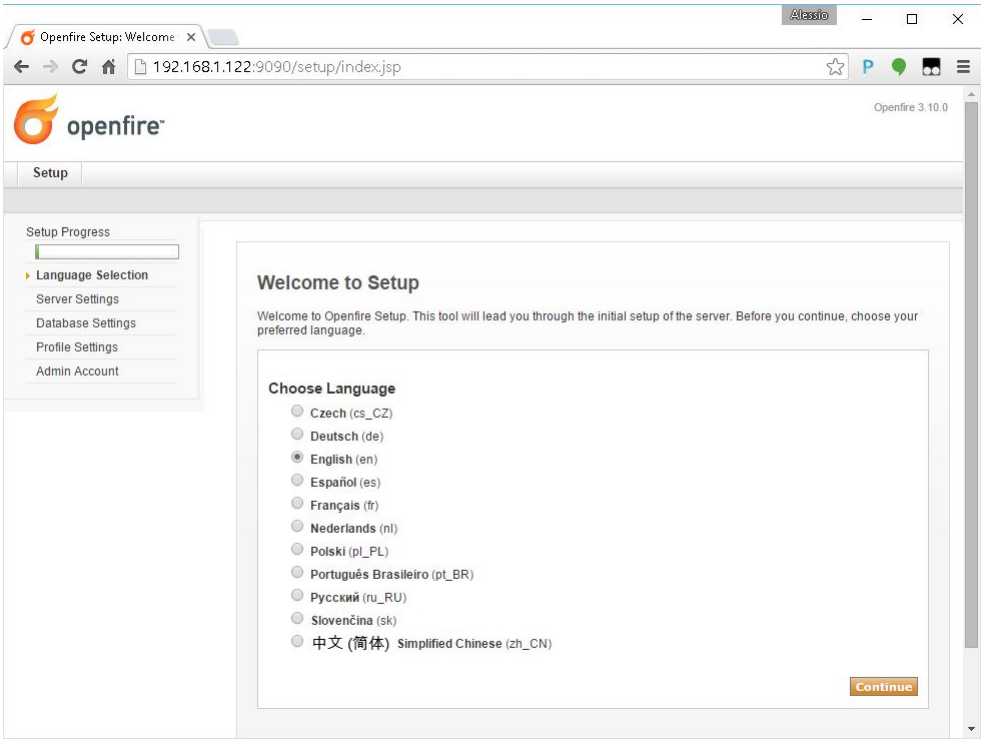
```
# as root

$> chkconfig openfire on

$> systemctl start openfire.service

# We need to open the firewall ports in order to expose the gui to the outside

$> firewall-cmd --permanent --zone=public --add-port=9090/tcp
$> firewall-cmd --permanent --zone=public --add-port=9091/tcp
$> firewall-cmd --reload
```

Configuration of Openfire server

Move the browser to the url

http://YOUR-SERVER-IP:9090

Choose preferable language and hit `Contine`



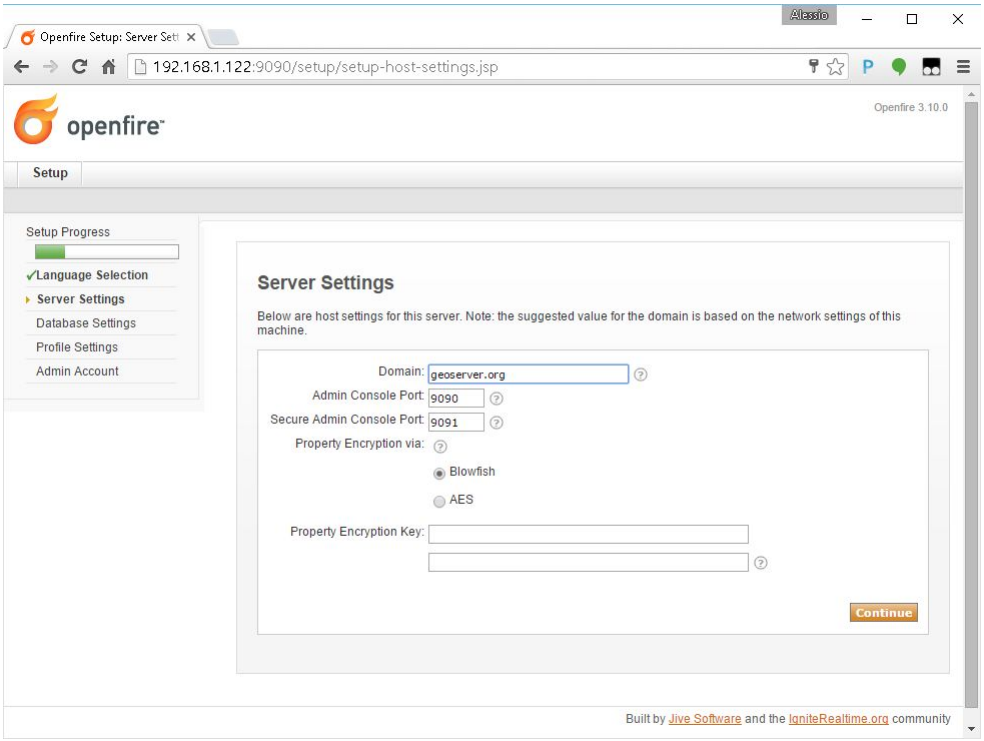
Specify the server Domain as

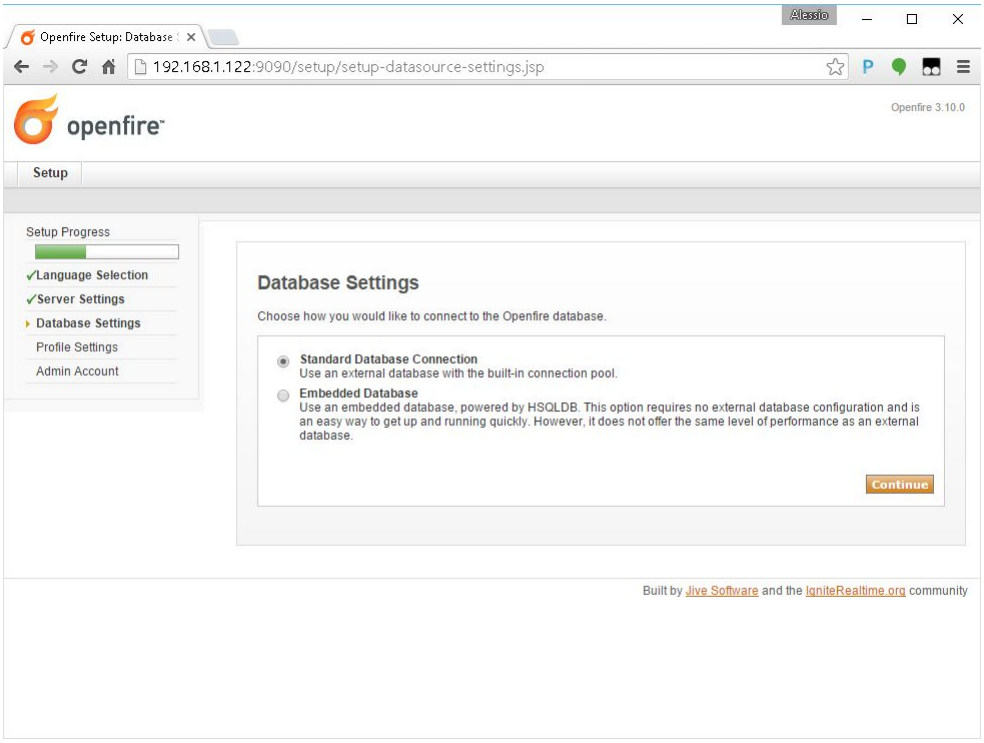
`geoserver.org`

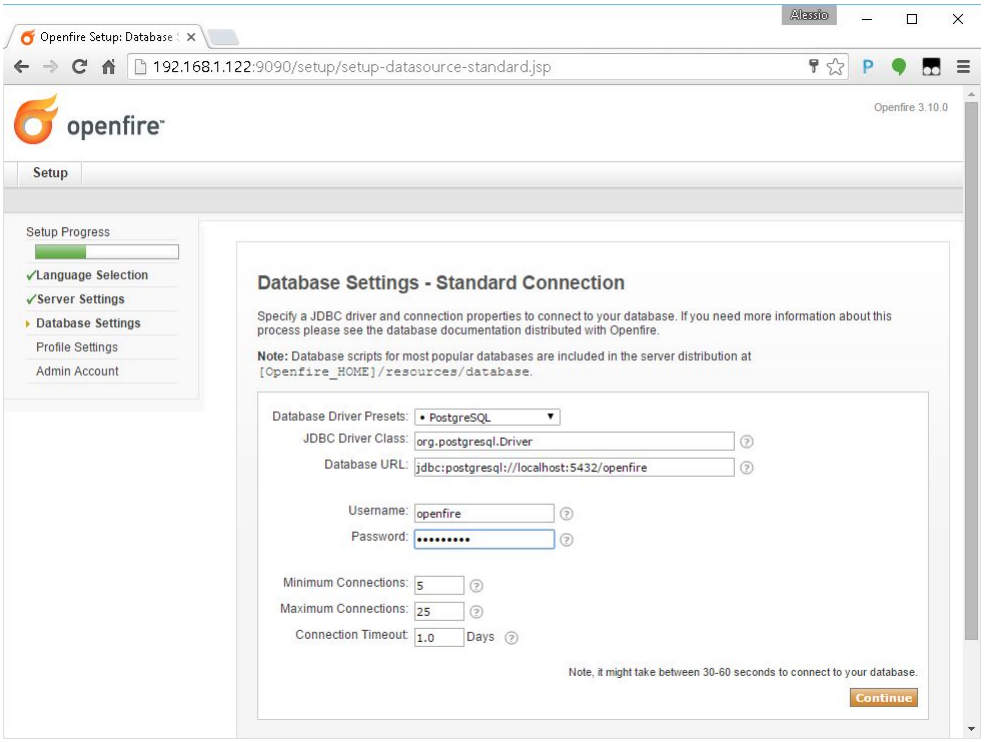
Choose the *Standard Database Connection* in the next section

Provide the Database connection parameters for the PostgreSQL DB in the standard connection section.

The password for the user `openfire` is the same provided in the PostgreSQL DB setup (see above).


Openfire Setup: Server Sett
192.168.1.122:9090/setup/setup-host-settings.jsp
openfire
Openfire 3.10.0
Setup
Setup Progress
✓Language Selection
Server Settings
Database Settings
Profile Settings
Admin Account
Server Settings
Below are host settings for this server. Note: the suggested value for the domain is based on the network settings of this machine.
Domain: geoserver.org
Admin Console Port: 9090
Secure Admin Console Port: 9091
Property Encryption via:
Blowfish
AES
Property Encryption Key:
Continue
Built by Jive Software and the IgniteRealtime.org community


Openfire Setup: Database
192.168.1.122:9090/setup/setup-datasource-settings.jsp
openfire
Openfire 3.10.0
Setup
Setup Progress
✓Language Selection
✓Server Settings
Database Settings
Profile Settings
Admin Account
Database Settings
Choose how you would like to connect to the Openfire database.
Standard Database Connection
Use an external database with the built-in connection pool.
Embedded Database
Use an embedded database, powered by HSQLDB. This option requires no external database configuration and is an easy way to get up and running quickly. However, it does not offer the same level of performance as an external database.
Continue
Built by Jive Software and the IgniteRealtime.org community



Note: Be sure the `openfire` database and user have been correctly created on PostgreSQL and the passwords provided (see above for instructions).

If there are no connection issues, choose `Default` value on the users profile settings section.

Create the *Administrator* account in the next section.

The password ***must*** match the one specified in the `remoteProcess.properties` file

```
R3m0T3wP5
```

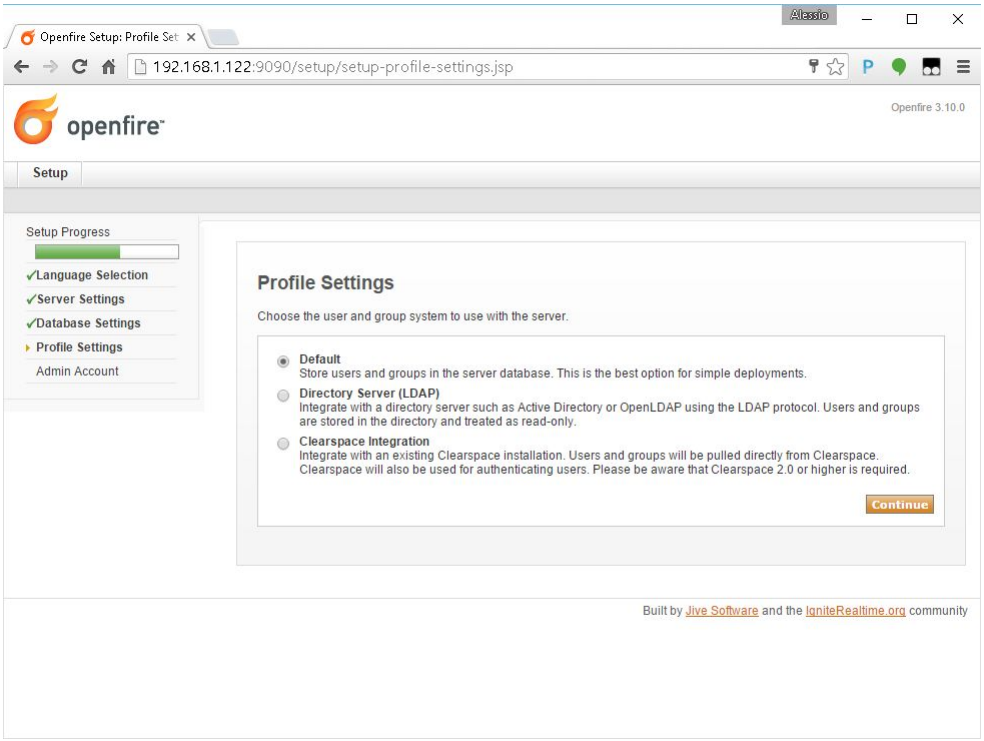
The initial setup is now complete. Log into the system using the newly created *admin* account.

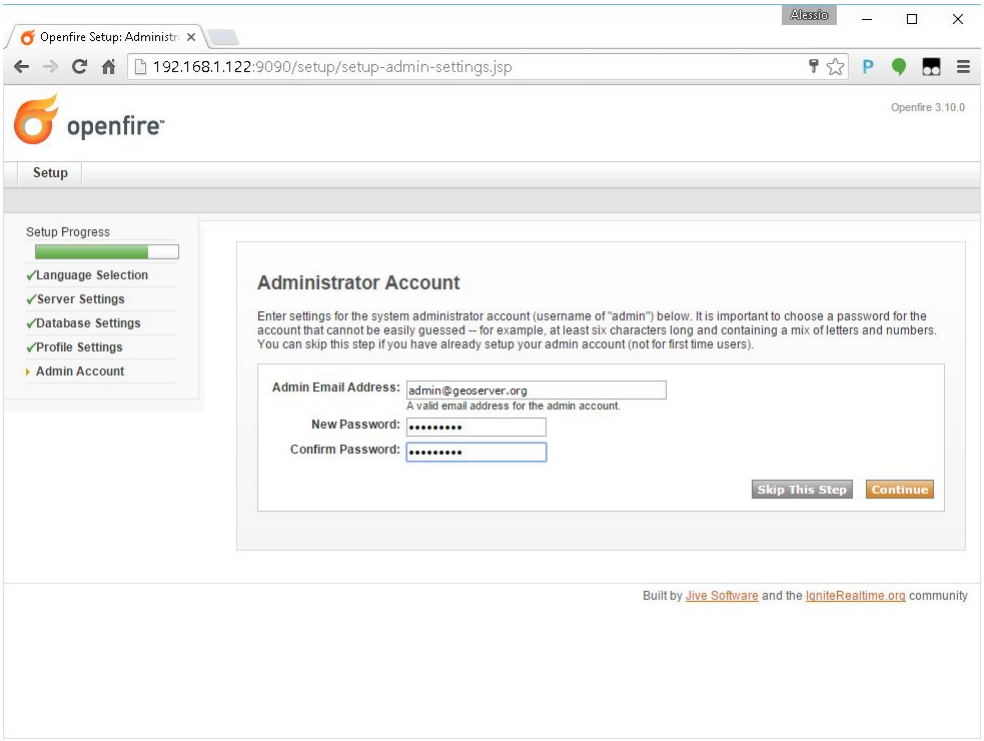
Move to the `Server Certificates` section of the `Server Settings` tab panel.

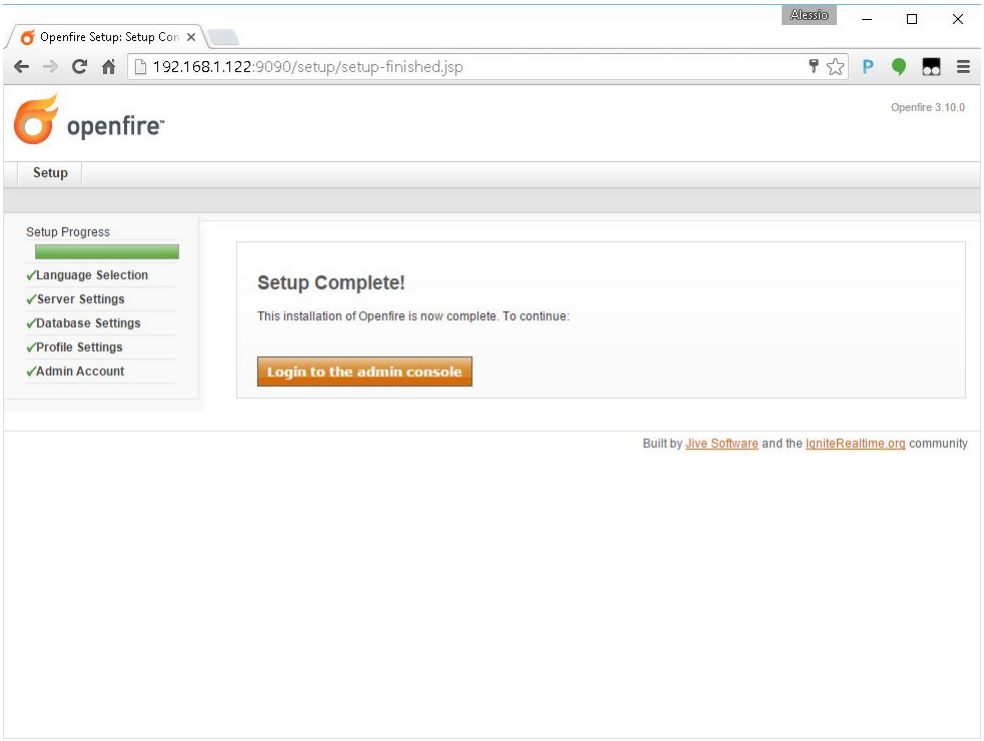
Warning: This passage is not needed anymnore on Openfire 4.0+. At least the management of the certificates is a bit different. Please refer to the specific Openfire documentation for more information.

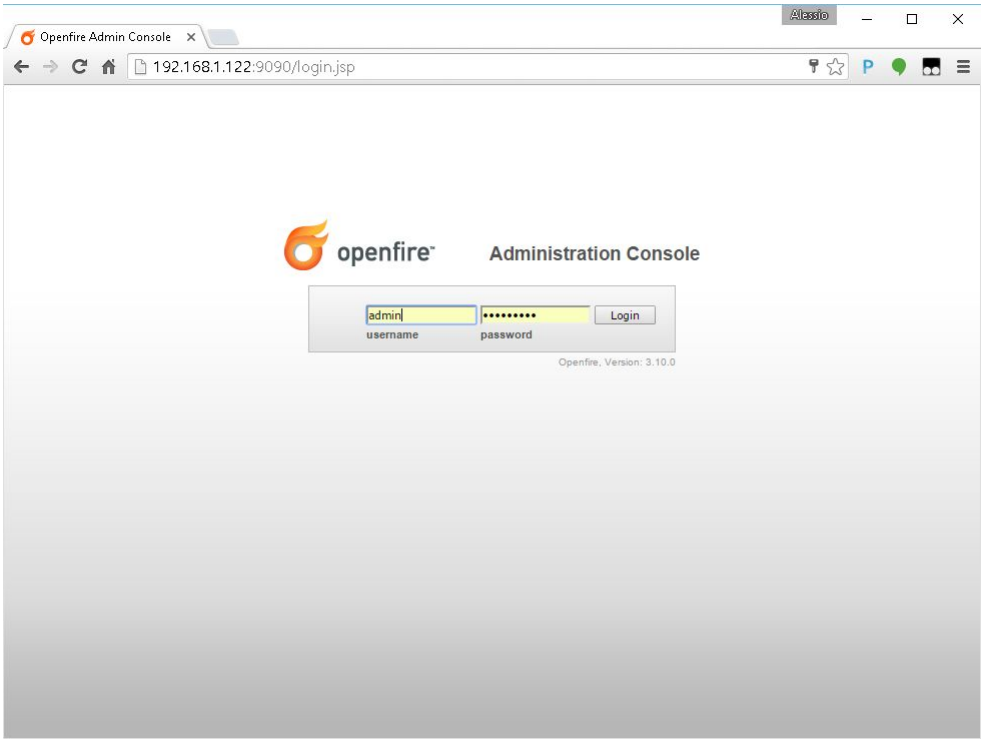
Make sure that the self-signed certificates have been correctly generated and click on `here` in order to restart the server

Warning: This passage is not needed anymnore on Openfire 4.0+. At least the management of the certificates is a bit different. Please refer to the specific Openfire documentation for more information.


Openfire Setup: Profile Set
192.168.1.122:9090/setup/setup-profile-settings.jsp
openfire
Openfire 3.10.0
Setup
Setup Progress
✓Language Selection
✓Server Settings
✓Database Settings
Profile Settings
Admin Account
Profile Settings
Choose the user and group system to use with the server.
Default
Store users and groups in the server database. This is the best option for simple deployments.
Directory Server (LDAP)
Integrate with a directory server such as Active Directory or OpenLDAP using the LDAP protocol. Users and groups are stored in the directory and treated as read-only.
Clearspace Integration
Integrate with an existing Clearspace installation. Users and groups will be pulled directly from Clearspace. Clearspace will also be used for authenticating users. Please be aware that Clearspace 2.0 or higher is required.
Continue
Built by Jive Software and the IgniteRealtime.org community


Openfire Setup: Administr
192.168.1.122:9090/setup/setup-admin-settings.jsp
openfire
Openfire 3.10.0
Setup
Setup Progress
✓Language Selection
✓Server Settings
✓Database Settings
✓Profile Settings
Admin Account
Administrator Account
Enter settings for the system administrator account (username of "admin") below. It is important to choose a password for the account that cannot be easily guessed -- for example, at least six characters long and containing a mix of letters and numbers. You can skip this step if you have already setup your admin account (not for first time users).
Admin Email Address:
admin@geoserver.org
A valid email address for the admin account.
New Password:
Confirm Password:
Skip This Step
Continue
Built by Jive Software and the IgniteRealtime.org community


Openfire Setup: Setup Con
192.168.1.122:9090/setup/setup-finished.jsp
openfire
Openfire 3.10.0
Setup
Setup Progress
✓Language Selection
✓Server Settings
✓Database Settings
✓Profile Settings
✓Admin Account
Setup Complete!
This installation of Openfire is now complete. To continue:
Login to the admin console
Built by Jive Software and the IgniteRealtime.org community

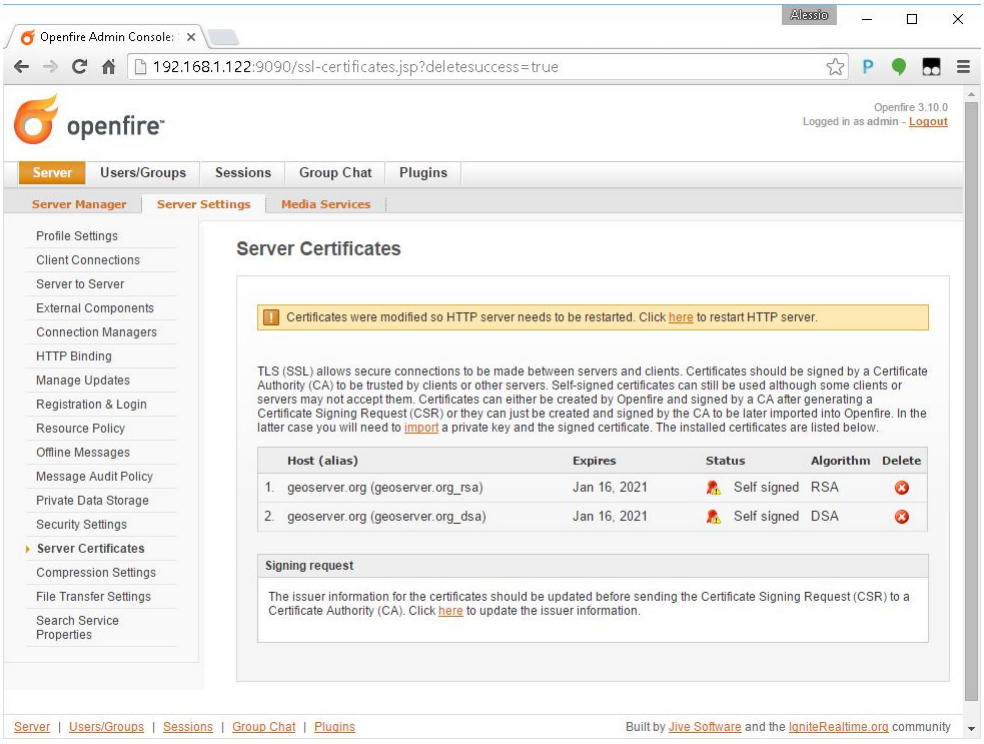

Openfire Admin Console
192.168.1.122:9090/login.jsp
openfire
Administration Console
admin
Login
username
password
Openfire, Version: 3.10.0

The same section now shows the server certificates and won't ask for another restart unless the certificates are generated again.

Update the `Security Settings` in order to allow the server accepting self-signed certificates on secured connections.

Warning: This passage is not needed anymnore on Openfire 4.0+. At least the management of the certificates is a bit different. Please refer to the specific Openfire documentation for more information.

Create the `default` channel as shown in the next figure.



Create the `management` channel as shown in the next figure. Pay attention to the `Room Options` and specify the password for the channel

```
R3m0T3wP5
```

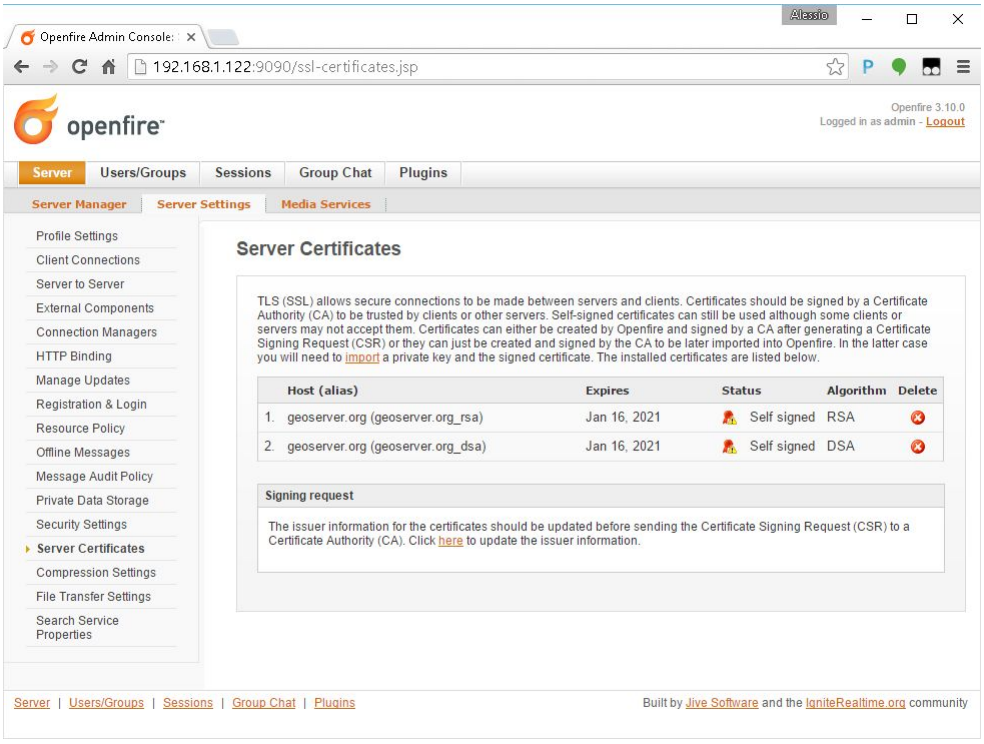
Double check that the channels have been correctly created and they appear in the `Group Chat Rooms`.

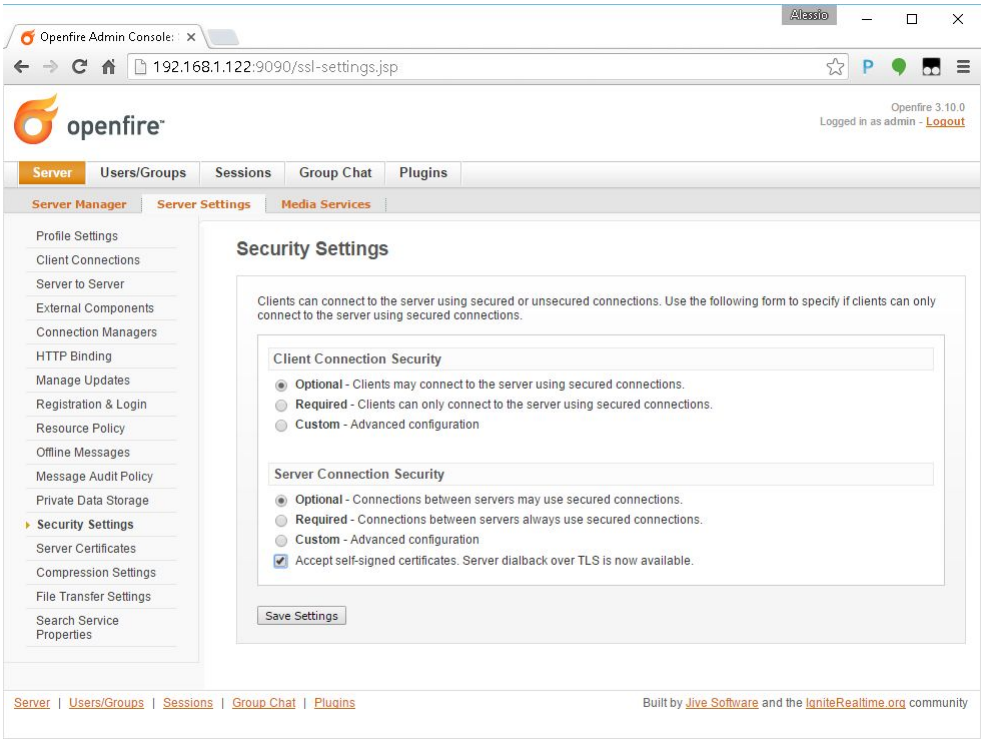
Restart GeoServer

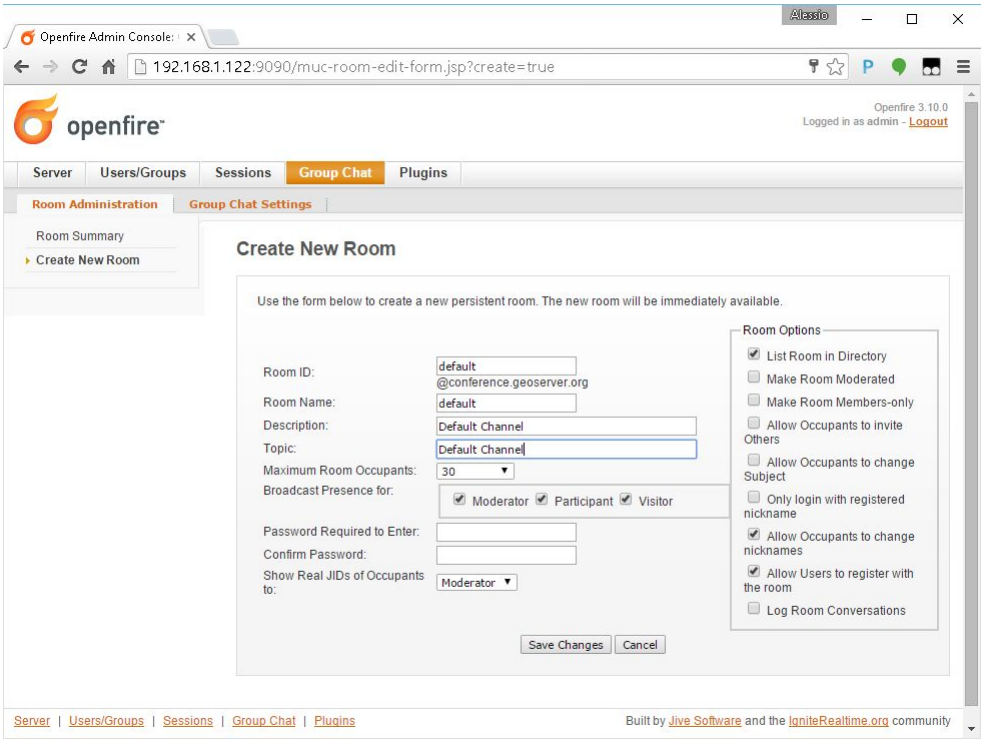
```
# as root

$> systemctl restart geoserver
```

After the GeoServer has successfully restarted, double check that it is connected to the server using the `admin` credentials.

It is ***very*** important that the user is shown as `Authenticated`.


Openfire Admin Console:
192.168.1.122:9090/ssl-certificates.jsp
openfire
Openfire 3.10.0
Logged in as admin - Logout
Server
Users/Groups
Sessions
Group Chat
Plugins
Server Manager
Server Settings
Media Services
Profile Settings
Client Connections
Server to Server
External Components
Connection Managers
HTTP Binding
Manage Updates
Registration & Login
Resource Policy
Offline Messages
Message Audit Policy
Private Data Storage
Security Settings
Server Certificates
Compression Settings
File Transfer Settings
Search Service Properties
Server Certificates
TLS (SSL) allows secure connections to be made between servers and clients. Certificates should be signed by a Certificate Authority (CA) to be trusted by clients or other servers. Self-signed certificates can still be used although some clients or servers may not accept them. Certificates can either be created by Openfire and signed by a CA after generating a Certificate Signing Request (CSR) or they can just be created and signed by the CA to be later imported into Openfire. In the latter case you will need to import a private key and the signed certificate. The installed certificates are listed below.
Host (alias)
Expires
Status
Algorithm
Delete
1. geoserver.org (geoserver.org_rsa)
Jan 16, 2021
Self signed
RSA
2. geoserver.org (geoserver.org_dsa)
Jan 16, 2021
Self signed
DSA
Signing request
The issuer information for the certificates should be updated before sending the Certificate Signing Request (CSR) to a Certificate Authority (CA). Click here to update the issuer information.
Server | Users/Groups | Sessions | Group Chat | Plugins
Built by Jive Software and the IgniteRealtime.org community


Openfire Admin Console:
192.168.1.122:9090/ssl-settings.jsp
openfire
Openfire 3.10.0
Logged in as admin - Logout
Server
Users/Groups
Sessions
Group Chat
Plugins
Server Manager
Server Settings
Media Services
Profile Settings
Client Connections
Server to Server
External Components
Connection Managers
HTTP Binding
Manage Updates
Registration & Login
Resource Policy
Offline Messages
Message Audit Policy
Private Data Storage
Security Settings
Server Certificates
Compression Settings
File Transfer Settings
Search Service Properties
Security Settings
Clients can connect to the server using secured or unsecured connections. Use the following form to specify if clients can only connect to the server using secured connections.
Client Connection Security
Optional - Clients may connect to the server using secured connections.
Required - Clients can only connect to the server using secured connections.
Custom - Advanced configuration
Server Connection Security
Optional - Connections between servers may use secured connections.
Required - Connections between servers always use secured connections.
Custom - Advanced configuration
Accept self-signed certificates. Server dialback over TLS is now available.
Save Settings
Server | Users/Groups | Sessions | Group Chat | Plugins
Built by Jive Software and the IgniteRealtime.org community



Check also that the user is registered to the XMPP channels created above.

Firewall Rules For XMPP Ports

By default the TCP Ports where the XMPP Server is listening for incoming connection are closed to the outside. Therefore it is necessary to enable the Firewall rules at least for the Openfire default secured port *5223* unless it has been changed by the user during the server setup.

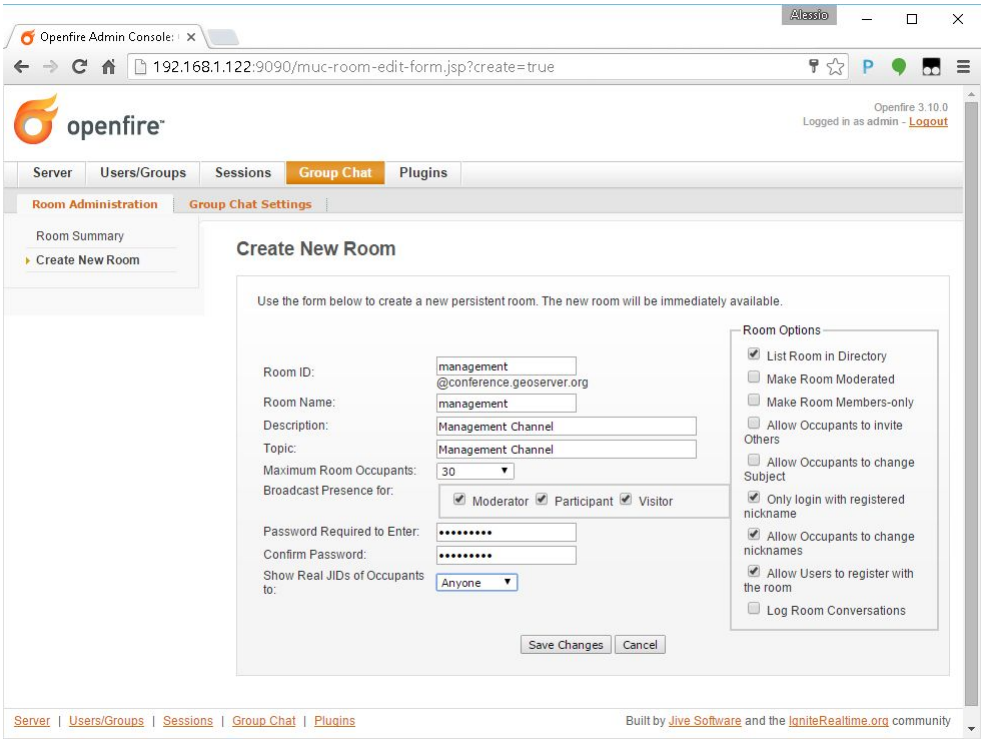
In order to do that issue the following commands:

```
# as root

# We need to open the firewall ports in order to expose the gui to the outside

$> firewall-cmd --permanent --zone=public --add-port=5222/tcp
$> firewall-cmd --permanent --zone=public --add-port=5223/tcp

$> firewall-cmd --reload
```

Forward Proxy to Apache HTTPD Server

The procedures described in this section allows to expose GeoServer via HTTPD through Apache HTTPD Server.

Those steps are not mandatory and the procedure may change accordingly to the final deployment on production systems.



In order to install Apache HTTPD Server proceed as follows:

```
# as root

$> yum -y install httpd mod_ssl

$> vi /etc/httpd/conf.d/forward-proxy.conf

   ProxyRequests Off

   ProxyPass /geoserver ajp://localhost:8009/geoserver
   ProxyPassReverse /geoserver ajp://localhost:8009/geoserver

$> systemctl enable httpd.service

$> service httpd restart
```

Selinux, enabled by default, needs to be instructed to allow http network connections. This can be done by running the command:

```
# as root

$> /usr/sbin/setsebool -P httpd_can_network_connect 1
```

Shared Folder through the NFS protocol

The next steps describe how to setup the system in order to expose a Shared Network Folder which will be used to store the outcomes of the remote processing.

The following procedures are not mandatory and the final deployment on the production system may be

configured to use different protocols and frameworks to expose shared file-systems.

The setup and initial configuration of the NFS packages can be done by following the next procedure:

```
# as root

$> yum -y install nfs-utils

$> vi /etc/idmapd.conf

   # The following should be set to the local NFSv4 domain name

   # The default is the host's DNS domain name.
   Domain = geoserver.org
```

Note: The domain specified above maybe different depending on the final system deployment and the production environment setup.

Creating and exposing a shared folder is possible by following the next steps:

1. as root
2. Create the physical folder structure to be exposed via the Network Filesystem

   ```
   $> mkdir /share
   $> mkdir /share/xmpp_data
   $> mkdir /share/xmpp_data/output
   $> mkdir /share/xmpp_data/resource_dir
   ```

3. Modify the rights in order to allow

   ```
   $> chmod -Rf 777 /share
   ```

3. Once the physical folder is ready it must be exposed via the `exports`

   ```
   $> vi /etc/exports
   ```

4. write settings for NFS exports

   ```
   /share host_ip/24(rw,no_root_squash)
   ```

4. Restart the NFS services

   ```
   $> systemctl start rpcbind nfs-server

   $> systemctl enable rpcbind nfs-server
   ```

Note: The **host_ip** must be the one of the host exposing the shared folder.

Selinux, enabled by default, needs to be instructed to allow NFS connections. This can be done by running the following commands:

```
# as root

$> setsebool -P httpd_use_nfs=1
```

```
$> setsebool -P samba_share_nfs=1

$> setsebool -P samba_export_all_ro=1

$> setsebool -P samba_export_all_rw=1
```

Deployment And Setup Of The XMPP Python Wrappers

Remote WPS Python Wrapper Framework

The Remote WPS Python framework source code is available on a public GitHub Repository of GeoSolutions S.A.S.

Users can install the "wps-remote" Python package by using the PyPi distribution

```
pip install wps-remote==2.11.0
```

The source code repository is available at the following address:

https://github.com/geoserver/wps-remote

The source code is available on the **master** branch.

How The System Works

This setup will configure the Remote WPS Python Wrapper to launch a Python executable called `test.py` that performs a `gdal_contour` on a GeoTIFF DEM.

The `test.py` executable takes in input just two parameters:

- `-i`; "–interval", nargs='?', default="10", help="Elevation interval between contours."
- `-w`; "–workdir", nargs='?', default="", help="Remote process sandbox working directory."

The paths of the command line and GeoTIFF to process (provided with the source code as sample data), are hard-coded into the Python code and must be changed accordingly to the system settings as explained later in the docs:

- `gdalContour = r'/usr/bin/gdal_contour'`
- `src = r'/share/xmpp_data/resource_dir/srtm_39_04/srtm_39_04_c.tif'`

The assumptions are that during the execution, the algorithms send logging and progress info to the standard output in a form similar to the following one:

```
2016-02-15 15:18:03,594 - main.create_logger - [INFO] ProgressInfo:100%
```

The log format has been configured through the `logger_test.properties` file:

```
[loggers]
keys=root

[handlers]
keys=consoleHandler

[formatters]
keys=simpleFormatter
```

```
[logger_root]
level=DEBUG
handlers=consoleHandler

[handler_consoleHandler]
class=StreamHandler
level=DEBUG
formatter=simpleFormatter
args=(sys.stdout,)

[formatter_simpleFormatter]
format=%(asctime)s - %(name)s - [%(levelname)s] %(message)s
datefmt=
```

The role of the Remote WPS Python wrapper is to take care of the communication between GeoServer WPS and the `test.py` executable.

The Python wrapper must be configured by specifying the number and type of input and output parameters of the executable, other than the connection parameter of the remote XMPP Server. The Python wrapper knows how to invoke the executables from the command line and how to parse and interpret the logging information thanks to some properties files containing a set of regular expressions which will be presented in details further in this document.

There must be a running instance of the Python wrapper for each executable, every one with its own specific configuration and XMPP user. The wrapper instances will connect automatically to the XMPP Server and GeoServer will send an **"invite"** message as soon it recognizes a new authenticated user appearing on the XMPP communication channels. In order to register the WPS Process into the GeoServer through the Remote Process Factory, the Python wrapper must reply to the invitation with a **"register"** message containing all the details about the I/O params of the executable. All those passages are managed automatically and transparently to the users by the Remote WPS Python framework.

Every time a user issues a new GeoServer WPS execute request, the Python wrapper starts a new thread calling the executables with the input parameters coming from GeoServer itself. The two running instances are connected through a unique "process execution ID" generated by GeoServer Remote Process Factory.

From now on, the Python wrapper thread follows the entire execution and takes care of sending feedbacks and logging information to the GeoServer Remote Process Factory, which are translated and forwarded to the GeoServer WPS Execution Manager. From the outside the users will experience a standard execution of an OGC WPS compliant process.

Summary Of The Configuration Steps

Connecting a new executable instance to GeoServer through the Python wrapper requires few configuration steps summarized here below:

1. **Clone a structure of `.properties` files containing:**
    - The connection parameter to the XMPP Server
    - The descriptor of the executable command line
    - The descriptor of the process I/O parameters
    - The logging informations
2. **Update the `remote.config` file with the correct XMPP Server information:**
    - Provide remote host and port parameters

- Provide domain and XMPP communication secured channels details
- Provide pointers to the shared folder

3. **Update the `logger.properties` file with the full path to the `service.log` file.**
4. **Update the `service.config` file with the executables parameters:**
   - The service name and the namespace

     Note: there must exist an user on the XMPP Server named as `namespace.serviceName` and a communication channel with the same identified of the service namespace.

     e.g.:

     – service = gdalContour

     – namespace = default

     means that on the XMPP Server we are looking for a communication channel named `default` and we will try to connect with the username `default.gdalContour`.

     Both of them must be defined before running the Python wrapper daemon.

   - The description of the service and the full path to the main executable
   - Other secondary parameters like the local output folder (where to store temporary results of the execution) and the max running time
   - The description of the Inputs and the actions to be taken
   - The description of the Outputs and the actions to be taken
   - The description of the logging information and the actions to be taken

Installation and Configuration Steps

Basic Environment Preparation

The following commands will prepare a MS Windows 7+, Windows 2008+ Server ISO machine for the deployment of:

1. Remote WPS Python wrapper
2. Sample configuration and testing of a sample executable `test.py` running the `gdal_contour` on a GeoTIFF DEM

Preparation of the system: standard and basic OS packages

Python

The system requires Python 2.7.9+ with few packages in order to work correctly. The installation of Python on a Windows system is quite fast

```
# as administrator

#.1 Download the Python 2.7.9 installation package from the browser, chosing the best␣
↪suitable distribution accordingly to the OS

  https://www.python.org/downloads/release/python-279/
```

```
#.2 Define the following System Environment Variables

PATH=%PATH%;C:\Python27;C:\Python27\Scripts
PYTHONPATH=.\;C:\Python27;C:\work\RemoteWPS
```

Other Mandatory Python Packages

```
# as administrator

# From a Command Line prompt

$> pip install wps-remote==2.11.0
```

Configure the RemoteWPS Environment

NFS Shared Folder

Link the shared folder to the `C:/share` through the NFS protocol. This is possible simply by turning on the NFS Services of the MS Windows functionalities and creating a client NFS connection to the NFS server.

Warning: "Services for NFS" have been removed on Windows 10. They are available only on Windows 10 Enterprise edition. For older Windows versions you can use the following procedure in order to enable NFS Client

Installing the client

1. Go to Control Panel → Programs → Programs and Features
2. Select: Turn Windows features on or off" from the left hand navigation.
3. Scroll down to "Services for NFS" and click the "plus" on the left
4. Check "Client for NFS"
5. Select "Ok"
6. Windows should install the client. Once the client package is install you will have the "mount" command available.

Mounting the export

This assumes the following:

- You know and can ping the hostname of the machine with the NFS exports
- The name of the exported filesystem (eg. /export, /home/users, /some/cool/file/path)
- The file systems are properly exported and accessible
  - Open a command prompt. (`Win` + `R`, enter "cmd" and press OK)
  - Type:

    ```
    mount \\{machinename}\{filesystem} {driveletter}
    ```

Examples:

```
mount \\filehost\home\users H:
mount \\server1234\long\term\file\storage S:
mount \\nas324\exports E:
```

Note: It is important that the shared folder structure is fully replicated on the Windows machine and the folder writable by the Windows processes.

```
| /share
|
|-- xmpp_data
|
|-- -- output
|
|-- -- resource_dir
```

First Deploy Of The RemoteWPS Python Framework

The wps-remote WHL archive contains a folder with a sample configuration

```
xmpp_data
```

Extract this folder and proceed with the next steps.

The files can also be downloaded from the GitHub source repository.

To clone the RemoteWPS Python Framework into a working folder, e.g.:

```
$> cd C:\work

$> git clone https://github.com/geoserver/wps-remote RemoteWPS
```

Setting Up The `remote.config`

```
# Edit the file xmpp_data/configs/remote.config

      [DEFAULT]

      bus_class_name = xmppBus.XMPPBus

      port = 5223
      address =  127.0.0.1
      domain = geoserver.org

      # . Those are the connection parameters to the XMPP Server.
      # . The user must exists on the Server and its name must be
      # . equal to the service name.
      user = default.GdalContour
      password = R3m0T3wP5

      mucService = conference.%(domain)s
      mucServicePassword = admin

      resource_file_dir = /share/xmpp_data/resource_dir

      # . Configure this option (along with 'backup_on_wps_execution_shared_dir'
      # . on single outputs of 'service.config') in order to make a copy
      # . of the results into a shared folder before sending messages to XMPP
      # . WARNING: this option takes precedence on "UPLOADER" option
      # wps_execution_shared_dir = /share/xmpp_data/share
```

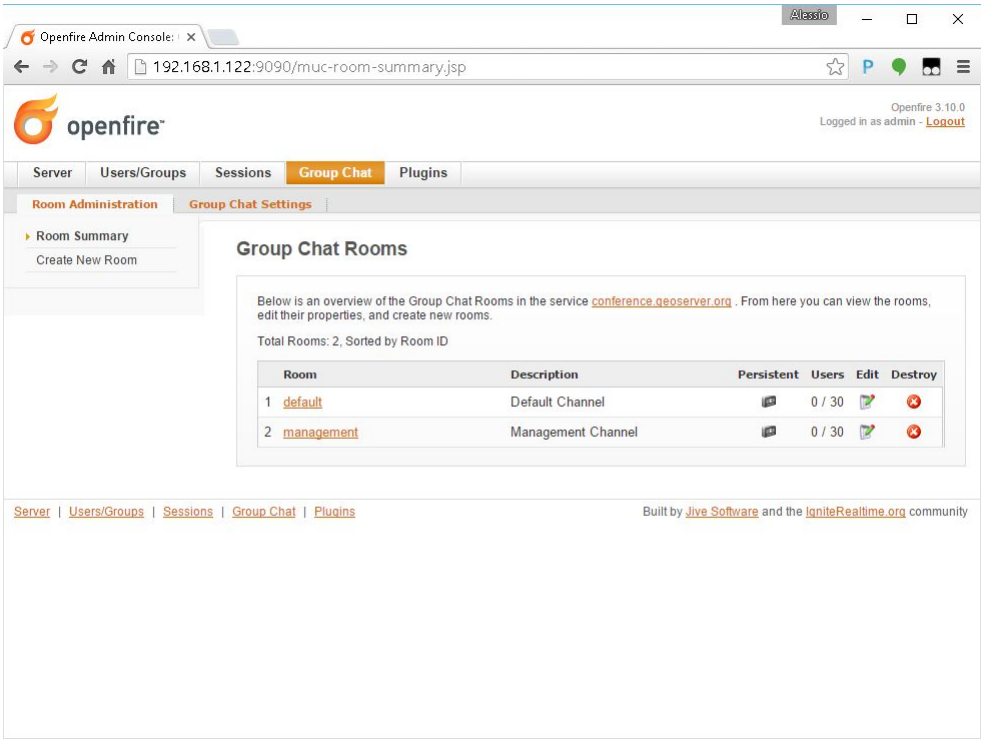
```
      # . This section is used to configure the uploader class and connection
      # . parameters.
      # . This is necessary in order to let the 'upload_data' option work on
      # . single outputs of 'service.config'
      [UPLOADER]
      # There are different implementations of the FTP Uploader available right now:
      # . Plain standard FTP Protocol (based on ftplib)
      #       ftpUpload.FtpUpload
      # . FTP over TLS (FTPS) Protocol (based on ftplib)
      #       ftpsUpload.FtpsUpload
      # . S-FTP Protocol (based on paramiko Python lib)
      #       sftpUpload.SFtpUpload
      uploader_class_name = ftpUpload.FtpUpload
      uploader_host = ftp.<your_host_here>:<your_port_here_default_21>
      uploader_username = <ftp_username>
      uploader_password = <ftp_password_encrypted>

      # . "encryptor" you can use encrypted passwords with a private/public key couple
      #
      # . To generate a new private key use the following command:
      #     openssl genrsa -out myTestKey.pem -passout pass:"f00bar" -des3 2048
      #
      # . To generate a new public key use the following command:
      #     openssl rsa -pubout -in myTestKey.pem -passin pass:"f00bar" -out␣
↪myTestKey.pub
      #
      # . To encrypt your password use the following utility
      #     python encrypt.py password path/to/rsakey.pub passphrase
      #
      # . To double check the password is correct use the following utility
      #     python decrypt.py password path/to/rsakey.pem passphrase
      uploader_private_rsa_key = /share/xmpp_data/ssl/myTestKey.pem
      uploader_passphrase = f00bar
```

The requisites for this configuration to work properly are:

1. Make sure the `<XMPP_server_ip_address>` is reachable and the port **5223** is allowed by the Firewall
2. Make sure the `default.GdalContour` user exists into the XMPP Server and that the password is correct
3. The MUC Service and the MUC Service Password are correct
4. The resource dir and the shared folder exists and are writable

Setting Up The `logger.properties`

```
# Edit the file xmpp_data/configs/logger.properties

[loggers]
keys=root

[handlers]
keys=consoleHandler,file

[formatters]
keys=simpleFormatter,consoleFormatter
```




```
[logger_root]
level=DEBUG
handlers=file, consoleHandler

[handler_consoleHandler]
class=StreamHandler
level=DEBUG
formatter=consoleFormatter
args=(sys.stdout,)
filter=

[handler_file]
class=handlers.TimedRotatingFileHandler
interval=midnight
backupCount=5
formatter=simpleFormatter
level=DEBUG
args=('/share/xmpp_data/service.log',)

[formatter_simpleFormatter]
format=%(asctime)s - %(name)s - %(levelname)s - %(message)s
datefmt=

[formatter_consoleFormatter]
format=%(asctime)s [%(levelname)s] %(message)s
datefmt=
```

The requisites for this configuration to work properly are:

1. Make sure the “C:/share/xmpp_data/” exists and is writable

Setting Up The `service.config`

```
# Edit the file xmpp_data/configs/myservice/service.config

      # This is a INI file to be read with python ConfigParser (https://docs.python.
↪org/2/library/configparser.html)
      # Is possible to reference another variable in the ini file using the format %(
↪<variable name>)s (note the 's' at the end)

      # ######################################### #
      # Default Service Params                    #
      # ######################################### #

      [DEFAULT]
      service = GdalContour
      namespace = default
      description = GDAL Contour Remote Service
      executable_path = /share/xmpp_data/configs/myservice/code
      executable_cmd = python %(executable_path)s/test.py
      output_dir = /share/xmpp_data/output/
      unique_execution_id = %(unique_exe_id)s
      workdir = %(output_dir)s/%(unique_execution_id)s
      active = True
      max_running_time_seconds = 300

      # . This option allows you to set the CPU and Memory average load scan time.
      # . It is espressed in 'minutes' and if disabled here it will be set by default
      # . to 15 minutes.
      load_average_scan_minutes = 1

      # . Use this option to completely avoid using this host (and prevent starting a␣
↪new
      # . 'processbot') whenever one of the following process names are running.
      # . In other words, if one of the following processes are currently running on␣
↪this machine,
      # . GeoServer won't send any WPS execute request until they are finished.
      process_blacklist = [resource consuming process name1, resource consuming␣
↪process name2]

      # ######################################### #
      # Inputs and Actions Declaration            #
      # ######################################### #

      [Input1]
      class = param
      name = interval
      title = Elevation Interval
      type = int
      description = Elevation interval between contours.
      min = 1
      max = 1
      default = 200

      [Action1]
      class = cmdline
      input_ref = interval
      alias = i
      template = -name value

      [Const1]
```

```
      class = const
      name = workdir
      type = string
      description = Remote process sandbox working directory
      value = %(workdir)s


      [Action2]
      class = cmdline
      input_ref = workdir
      alias = w
      template = -name value

      # ########################################## #
      # Output Parameters Declaration              #
      # ########################################## #

      [Output1]
      name = result1
      type = application/zip
      description = WPS Resource Binary File
      title = SRTM
      filepath = %(workdir)s/contour.zip
      publish_as_layer = true
      publish_default_style = polygon
      publish_target_workspace = it.geosolutions
      publish_layer_name = contour

      # . Enable this option in order to perform a backup of this output
      # . before sending it to GeoServer.
      # . WARNING: This option works only along with 'wps_execution_shared_dir'
      # .          option on 'remote.config', and takes precedence on 'upload_data'
      # backup_on_wps_execution_shared_dir = true

      # . Enable this option if you want the output to be uploaded on remote host.
      # . Notice that you must also configure uploader parameters on 'remote.config'
      # upload_data = true

      # . Optionally it is possible to specify a root folder if the uploader class␣
↪supports it.
      # upload_data_root = /remote-wps/default

      [Output2]
      name = result2
      type = application/x-netcdf
      description = NetCDF Binary File
      title = flexpart
      filepath = %(output_dir)s/flexpart.nc
      publish_as_layer = true
      publish_default_style = raster
      publish_target_workspace = it.geosolutions
      publish_layer_name = flexpart

      # . Enable this option in order to perform a backup of this output
      # . before sending it to GeoServer.
      # . WARNING: This option works only along with 'wps_execution_shared_dir'
      # .          option on 'remote.config', and takes precedence on 'upload_data'
      # backup_on_wps_execution_shared_dir = true
```

```
      # . Enable this option if you want the output to be uploaded on remote host.
      # . Notice that you must also configure uploader parameters on 'remote.config'
      # upload_data = true

      # . Optionally it is possible to specify a root folder if the uploader class␣
↪supports it.
      # upload_data_root = /remote-wps/default

      [Output3]
      name = result3
      type = application/owc
      description = WPS OWC Json MapContext
      layers_to_publish = result2
      publish_as_layer = true
      publish_layer_name = owc_json_ctx
      publish_metadata = /share/xmpp_data/resource_dir/owc_json_ctx.json

      # ######################################### #
      # Logging Options Declaration               #
      # ######################################### #

      [Logging]
      stdout_parser = [.*\[DEBUG\](.*), .*\[INFO\] ProgressInfo\:([-+]?[0-9]*\.?[0-
↪9]*)\%, .*\[(INFO)\](.*), .*\[(WARN)\](.*), .*\[(ERROR)\](.*), .*\[(CRITICAL)\](.*)]
      stdout_action = [ignore,          progress,                                  ␣
↪     log,              log,              abort,               abort]
```

The requisites for this configuration to work properly are:

1. Make sure the `default.GdalContour` user exists into the XMPP Server and that the password is correct
2. Make sure the `default` channel exists on the XMPP Server
3. Make sure the executable path and command are correct
4. Make sure the `output_dir` exists and is writable
5. Make sure the `max_running_time_seconds` have been set to a value high enough to allow the executables to complete the jobs.

   The GeoServer instance must also respect the WPS execution timings which must be configured accordingly. In order to do that access to the GeoServer Web Admin GUI.

   http://host:8080/geoserver/web/

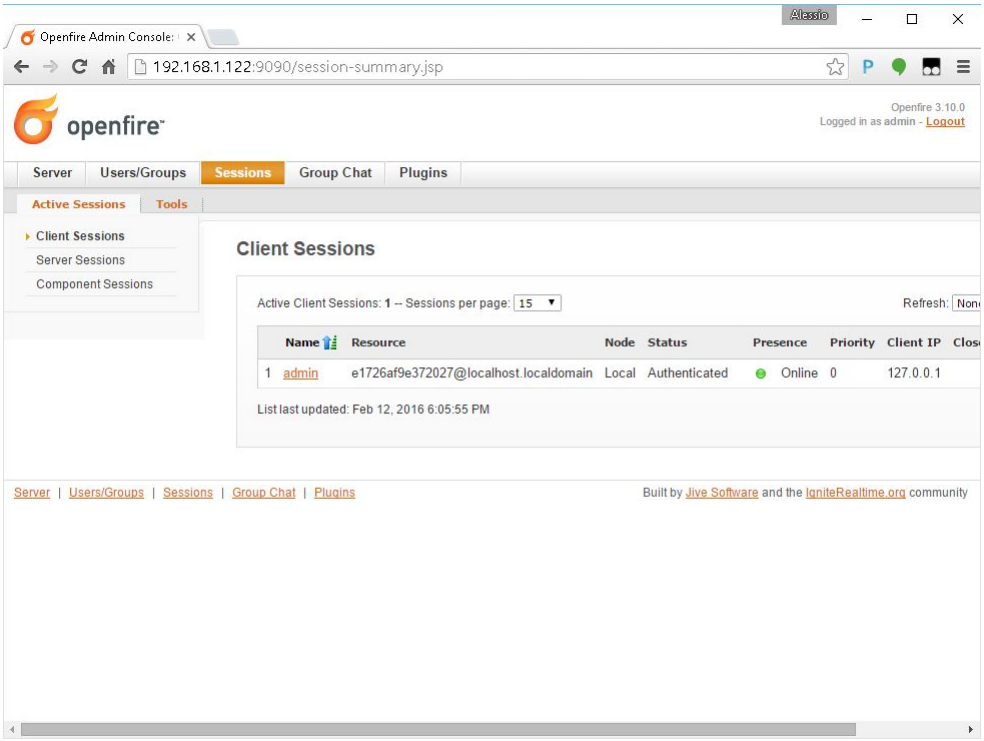
   login as administrator (default credentials are admin/geoserver which should be changed anyway).

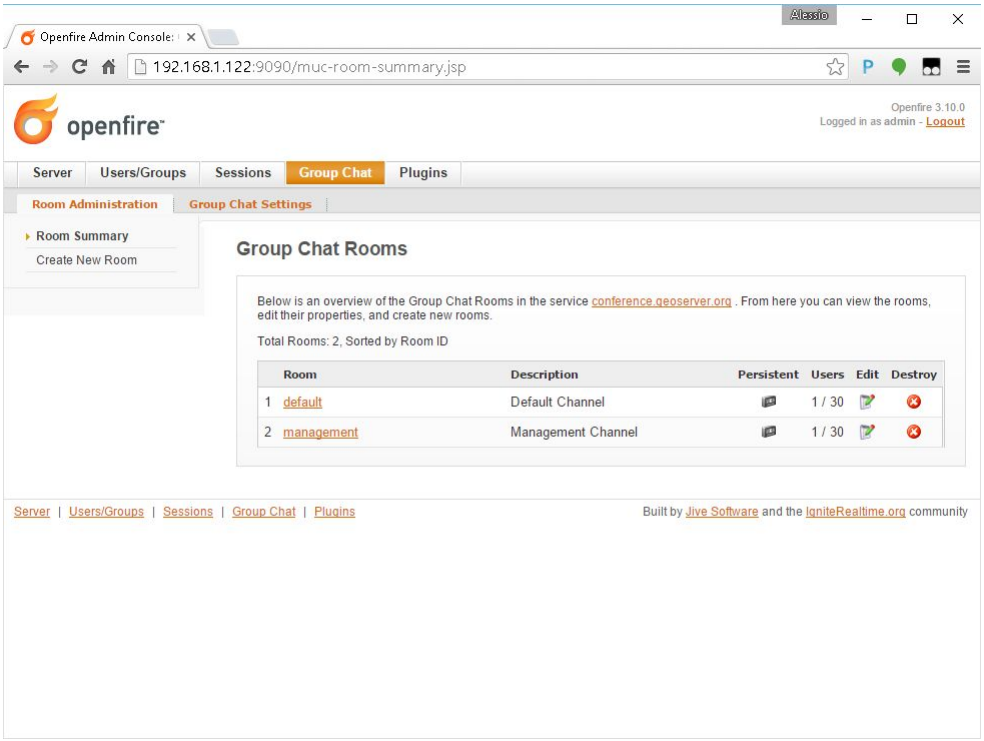
   From the Web Processing Service settings page

   The timeouts and the number of parallel executions (both async and sync) must be tuned accordingly to the execution needs.

6. Make sure the inputs have been configured correctly for the command line execution

   ```
   [Input1]
   class = param
   name = interval
   title = Elevation Interval
   type = int
   description = Elevation interval between contours.
   ```


Openfire Admin Console:
192.168.1.122:9090/session-summary.jsp
openfire
Openfire 3.10.0
Logged in as admin - Logout
Server
Users/Groups
Sessions
Group Chat
Plugins
Active Sessions
Tools
Client Sessions
Server Sessions
Component Sessions
Client Sessions
Active Client Sessions: 1 -- Sessions per page: 15
Refresh:
Name
Resource
Node
Status
Presence
Priority
Client IP
1 admin e1726af9e372027@localhost.localdomain Local Authenticated Online 0 127.0.0.1
List last updated: Feb 12, 2016 6:05:55 PM
Server | Users/Groups | Sessions | Group Chat | Plugins
Built by Jive Software and the IgniteRealtime.org community


Openfire Admin Console:
192.168.1.122:9090/muc-room-summary.jsp
openfire
Openfire 3.10.0
Logged in as admin - Logout
Server
Users/Groups
Sessions
Group Chat
Plugins
Room Administration
Group Chat Settings
Room Summary
Create New Room
Group Chat Rooms
Below is an overview of the Group Chat Rooms in the service conference.geoserver.org . From here you can view the rooms, edit their properties, and create new rooms.
Total Rooms: 2, Sorted by Room ID
Room
Description
Persistent
Users
Edit
Destroy
1 default Default Channel 1 / 30
2 management Management Channel 1 / 30
Server | Users/Groups | Sessions | Group Chat | Plugins
Built by Jive Software and the IgniteRealtime.org community

```
min = 1
max = 1
default = 200

[Action1]
class = cmdline
input_ref = interval
alias = i
template = -name value
```

The configuration above sets an input of type `int` (the expected value will be interpreted as text and declared as Literal to the WPS), which is mandatory (**min = 1**) and can have a single value (**max = 1**).

The `[Action1]` is connected to the input through the `input_ref` which is equal to the `[Input1].name`.

In the example above the action simply gets the input value specified by the user and forward it to the command line.

The final result will be something lihe this:

```
$> /work/RemoteWPS/xmpp_data/configs/myservice/code/test.py <input_value_here>
```

The `[Action1].template` property allows to specify the name of the option if required by the executable.

As an instance the following value for the `[Action1].template`:

```
alias = i
template = -name value
```

will result in something like this:

```
$> /work/RemoteWPS/xmpp_data/configs/myservice/code/test.py -i <input_value>
```

There exists other types of input and actions.

As an instance it is possible to specify `constant` input types like the following one:

```
[Const1]
class = const
name = workdir
type = string
description = Remote process sandbox working directory
value = %(workdir)s

[Action2]
class = cmdline
input_ref = workdir
alias = w
template = -name value
```

The `[Const1].value` can be a constant value or a reference to the configuration file properties.

In the example above we are going to pass to the command line the full path of the process workind directory, which is a unique folder created at runtime where the RemoteWPS framework stores temporary and intermediate results of the process execution.

Enabling the constant input above, the resulting command line will be something like the following one:

```
$> /work/RemoteWPS/xmpp_data/configs/myservice/code/test.py -i <input_value> -w /
↪share/xmpp_data/output/<exec_id>
```

Note: The **<exec_id>** is known at runtime only.

7. Make sure the outputs have been configured correctly for the command line execution

```
[Output1]
name = result1
type = application/zip
description = WPS Resource Binary File
title = SRTM
filepath = %(workdir)s/contour.zip
publish_as_layer = true
publish_default_style = polygon
publish_target_workspace = it.geosolutions
publish_layer_name = contour
```

In the example above we declare to the WPS only **one** output of type `application/zip`.

In this case the RemoteWPS framework expects to find a `contour.zip` file at the end of the execution into the working directory (see above).

There are many kind of possible outputs which can be defined here. As an instance it is possible to define an output of type `string` which can read the outcome from a file and stream it out as plain text.

It is also possible to define several kind of binary outputs depending on the executable outcomes. For more details please refer to the Remote WPS Python framework specific documentation at the end of this section.

8. Make sure the regular expressions of the "stdout_parser" are correct and valid accordingly to the output of the executable

```
[Logging]
stdout_parser = [.*\[DEBUG\](.*), .*\[INFO\] ProgressInfo\:([-+]?[0-9]*\.?[0-9]*)\
↪%, .*\[(INFO)\](.*), .*\[(WARN)\](.*), .*\[(ERROR)\](.*), .*\[(CRITICAL)\](.*)]
stdout_action = [ignore,          progress,                                    ␣
↪   log,              log,              log,               abort]
```

The example configuration above:

- Ignores all `STDOUT` debug logs received from `test.py`
- Translates **as** *progress info message* any number parsed by the regex from `STDOUT` and sends it to GeoServer WPS.
- Logs all `STDOUT` info, warn and error logs received from `test.py`
- Translates **as** *abort message* any keyword **CRITICAL** parsed by the regex from `STDOUT` and sends it to GeoServer WPS.

At least **progress** and **abort** messages are mandatory in order to take track of the process execution progress and fault state.

A Running Example

In the section *A Remote "Gdal Contour" Process Binding Example* will show how to run the example and how to parse the results in GeoServer.

ANNEX A: Remote WPS Python Wrapper Reference

This section is meant to be a summary of the current possible options for the RemoteWPS Python Wrapper `service.config` configuration.

Default Section

```
# ########################################### #
# Default Service Params                      #
# ########################################### #

[DEFAULT]
service = GdalContour
namespace = default
description = GDAL Contour Remote Service
executable_path = /work/RemoteWPS/xmpp_data/configs/myservice/code
executable_cmd = python %(executable_path)s/test.py
output_dir = /share/xmpp_data/output/
unique_execution_id = %(unique_exe_id)s
workdir = %(output_dir)s/%(unique_execution_id)s
sharedir = /home/myproc/repository/default
active = True
max_running_time_seconds = 300
load_average_scan_minutes = 1
process_blacklist = [resource consuming process name1, resource consuming process␣
↪name2]
```

- **service**; The name of the WPS service. On GeoServer the WPS Process will be represented as `namespace.service`

  Note: The XMPP Server *must* have a registered user named like the fully qualified service name `namespace.service`

- **namespace**; The namespace (or prefix) of the service. Along with the *service* parameter, it represents the fully qualified name of the service.
- **description**; This contains the textual description of the GeoServer WPS Process.
- **executable_path**; Full path of the executable to wrap.
- **executable_cmd**; Executable command.
- **output_dir**; The base output folder where the Python wrapper stores logs and temporary files.
- **unique_execution_id**; The unique ID generated by GeoServer and sent to the process via the *REQUEST* command message.
- **workdir**; Temporary folder where to store the outcomes and log files.
- **sharedir**; Sahred folder where to **backup** outcomes with `backup_on_wps_execution_shared_dir` property equal *true*

- **active**; *Boolean* which enables or disables the service.
- **max_running_time_seconds**; After this time the Python Wrapper tries to shutdown the process and send a *FAILED* message to GeoServer.
- **load_average_scan_minutes**; This option allows you to set the CPU and Memory average load scan time. It is espressed in 'minutes' and if disabled here it will be set by default to 15 minutes.
- **process_blacklist**; Use this option to completely avoid using this host (and prevent starting a new 'processbot') whenever one of the following process names are running. In other words, if one of the following processes are currently running on this machine, GeoServer won't send any WPS execute request until they are finished.

Inputs Section

```
# ######################################### #
# Inputs and Actions Declaration            #
# ######################################### #

[Input1]
class = param
name = interval
title = Elevation Interval
type = int
description = Elevation interval between contours.
min = 1
max = 1
default = 200

[Action1]
class = cmdline
input_ref = interval
alias = i
template = -name value

[Const1]
class = const
name = workdir
type = string
description = Remote process sandbox working directory
value = %(workdir)s

[Action2]
class = cmdline
input_ref = workdir
alias = w
template = -name value
```

The *Inputs Section* can contain three type of objects:

1. `[Input#]`; Descriptor of the corresponding GeoServer WPS Input parameter.
2. `[Action#]`; `1..n` actions of the Python Wrapper associated to an `[Input]`. The reference is done through the **input_ref** property.
3. `[Const#]`; Constant values passed to the executable and transparent to the GeoServer WPS.

[Input#]

- **class**; Uses introspection to instantiate an Input parameter. Currently the only value admitted is `param`
- **name**; The name of the input parameter. This will be also the name of the GeoServer Input parameter.
- **title**; The title of the input parameter. To be used as internal descriptor.
- **description**; The description of the input parameter. This will be also the description of the GeoServer Input parameter.
- **type**; The type of the input parameter. Allowed types are:
  1. `string`; Simple text input. Invalid characters will be automatically removed.
  2. `int`; Integer numeric input value.
  3. `float`; Float numeric input value.
  4. `url`; Must contain a valid URL. Invalid characters will be automatically removed.
  5. `application/json`; Threated as a JSON string. It will be parsed by the Python Wrapper and converted into a complex object.
  6. `datetime`; Converted into a Python `datetime` object accordingly to the **formatter** property containing the date pattern and which must also be provided.
- **min**; Optional parameter which sets the minimum set of inputs of this type allowed by the GeoServer WPS. *0* by default.
- **max**; Optional parameter which sets the maximum set of inputs of this type allowed by the GeoServer WPS. *0* (alias infinite) by default.
- **default**; Optional parameter for setting the default value of this input if a value has not provided.
- **formatter**; Optional parameter to be used along with `datetime` inputs. Defines the date pattern to be applied to the input string (e.g.: %Y-%m-%d %H:%M:%S)

[Action#]

- **class**; Uses introspection to instantiate the type of Action.
  1. `cmdline`; The value of the associated input will be passed to the executable as a key-value pair accordingly to the `template` specified (e.g.: –name=value).
     - `template`; Template of the key-value pair format (e.g.: template = -name value)
     - `alias`; Alias of the key (e.g.: alias = i will be translated as -i value)
  2. `createJSONfile`; The value of the associated input will be dumped to a JSON file and the reference to the file passed to the executable.
     - `target_filepath`; PATH Where to store the JSON file.
     - `json_schema`; The PATH to the JSON Schema to be used to validate the input values.
  3. `updateJSONfile`; The value of the associated input will be substituted into a target template JSON file, which then will be passed to the executbale as reference.
     - `source_filepath`; PATH of the source JSON template file.
     - `target_filepath`; PATH of the target JSON file.
     - `json_path_expr`; JSON path expression used to subsitute the values.
  4. `copyfile`; The value of the associated input will be interpreted as a path to a source file. The content of the file will be copied into a temporary file and then passed to the executbale as reference.

  - `source_filepath`; PATH of the source JSON template file.
  - `target_filepath`; PATH of the target JSON file.

5. `updateINIfile`; The value of the associated input will be substituted into a target template INI file, which then will be passed to the executbale as reference.
   - `source_filepath`; PATH of the source JSON template file.
   - `target_filepath`; PATH of the target JSON file.
   - `section`; Section of the INI file where to store key-value pair entries.
6. `updateINIfileList`; The value of the associated input will be parsed as a `list` and substituted into a target template INI file, which then will be passed to the executbale as reference.
   - `source_filepath`; PATH of the source JSON template file.
   - `target_filepath`; PATH of the target JSON file.
   - `section`; Section of the INI file where to store key-value pair entries.

- **input_ref**; `name` of the input parameter referenced by this Action.

[Const#]

- **class = const**
- **name**; Name of the input parameter, used by an action as reference.
- **type**; May be one of the **[Input#].type** ones.
- **description**; Internal description of the parameter.
- **value**; Fixed value parsed by the referencing Action.

Outputs Section

```
# ########################################## #
# Output Parameters Declaration              #
# ########################################## #

# WARNING: the name must start with the keyword "result"

[Output1]
name = result1
type = string
description = WPS Resource Plain Text
filepath = %(workdir)s/geoserverLayerOutput.xml

[Output2]
name = result2
type = image/geotiff
description = WPS Resource Binary File
title = SRTM
filepath = %(workdir)s/srtm_39_04_c.tif
backup_on_wps_execution_shared_dir = true
publish_as_layer = true
publish_default_style = raster
publish_target_workspace = it.geosolutions
publish_layer_name = srtm_39_04_c
# Such metadata is a JSON snippet itslef (/tmp/resource_dir/result2.json) with a␣
↪small particularity.
```

```
# Since you cannot know a-priori some of the final Layer properties,
# you can use inside the json (/tmp/resource_dir/result2.json) some keywords which␣
↪will be updated
# automatically by the RemoteWPS which are the following ones:
#
# ${type}
# ${name}
# ${title}
# ${description}
# ${lastUpdated}
# ${getMapBaseUrl}
# ${srs}
# ${bbox}
# ${workspace}
# ${layers}
# ${styles}
publish_metadata = /<path_to>/resource_dir/result2.json

[Output3]
name = result3
type = image/geotiff;stream
description = WPS Resource Binary Stream
title = This Is A GeoTIFF Layer
filepath = %(workdir)s/srtm_39_04_c.tif
publish_as_layer = true
publish_default_style = raster
publish_target_workspace = it.geosolutions
publish_layer_name = srtm_39_04_c

[Output4]
name = result4
type = application/x-netcdf
description = NetCDF Binary File
title = Wind
filepath = %(workdir)s/RS1_STB_1FSCLS20111003_175545_00000018xS2x_16bxx_83066_29447_
↪wind.nc
backup_on_wps_execution_shared_dir = true
publish_as_layer = true
publish_default_style = raster
publish_target_workspace = it.geosolutions
publish_layer_name = wind
# Such metadata is a JSON snippet itslef (/tmp/resource_dir/result3.json) with a␣
↪small particularity.
# Since you cannot know a-priori some of the final Layer properties,
# you can use inside the json (/tmp/resource_dir/result4.json) some keywords which␣
↪will be updated
# automatically by the RemoteWPS which are the following ones:
#
# ${type}
# ${name}
# ${title}
# ${description}
# ${lastUpdated}
# ${getMapBaseUrl}
# ${srs}
# ${bbox}
# ${workspace}
# ${layers}
```

```
# ${styles}
publish_metadata = /<path_to>/resource_dir/result4.json

# ######################################### #
# GML Possible type values are              #
#  text/xml;subtype=gml/3.1.1               #
#  text/xml;subtype=gml/2.1.2               #
#  application/gml-3.1.1                    #
#  application/gml-2.1.2                    #
# ######################################### #
[Output5]
name = result5
type = text/xml;subtype=gml/3.1.1
description = WPS Resource GML
filepath = %(workdir)s/geoserverLoadLayerOutput.xml

[Output6]
name = result6
type = video/mp4
description = Video MP4 Binary File
title = Wind
filepath = %(workdir)s/RS1_STB_1FSCLS20111003_175545_00000018xS2x_16bxx_83066_29447_
↪wind.mp4
backup_on_wps_execution_shared_dir = false

[Output7]
name = result7
type = application/owc
description = WPS OWC Json MapContext
layers_to_publish = result2;result4
publish_as_layer = true
publish_layer_name = owc_json_ctx
publish_metadata = /<path_to>/resource_dir/owc_json_ctx.json
```

The examples above represents all the possible types of Outputs currently supported by the Remote WPS Wrapper.

- **type = string**

  The content of the file specified by the `filepath` is read and sent to the WPS. The GeoServer WPS declares this as a `text/plain` output type.

- **type = image/geotiff**

  The content of the binary GeoTIFF specified by the `filepath` is read and sent to the WPS. The GeoServer WPS declares this as a `otput binary` RAW FILE output type.

- **type = image/geotiff;stream**

  The content of the binary GeoTIFF specified by the `filepath` is read and sent to the WPS. The GeoServer WPS declares this as a `otput binary` RAW STREAM output type.

- **type = application/x-netcdf**

  The content of the binary NetCDF specified by the `filepath` is read and sent to the WPS. The GeoServer WPS declares this as a `otput binary` RAW FILE output type.

- **type = text/xml;subtype=gml/3.1.1**

  The content of the file specified by the `filepath` is read and sent to the WPS. The GeoServer WPS declares this as a `text/xml` output type.

- **type = video/mp4**

  The content of the binary MPEG-4 specified by the `filepath` is read and sent to the WPS. The GeoServer WPS declares this as a `otput binary` RAW FILE output type.

- **type = application/owc**

  This is a particular type of output. From the GeoServer WPS point of view is a `text/plain` JSON output type describing a Web Mapping Context.

  The Remote WPS Plugin on GeoServer side takes care of publishing the layers specified by `layers_to_publish = result2;result4` and render the templates specified by `publish_metadata` of each output.

  The outcome will be a complex JSON WMC describing the map to publish.

  In order to activate this funcionality, update the GeoServer `remoteProcess.properties` on the `GEOSERVER_DATA_DIR` with a new option:

```
# full path to the template used to generate the OWS WMC Json output

owc_wms_json_template = /tmp/resource_dir/wmc_template.json
```

Sample `wmc_template.json`

```
{
  "type": "FeatureCollection",
  "id": "GeoServer OWC Map Context: version of 2015-07-14",
  "geometry": {
              "type":"Polygon",
              "coordinates": ${renderingArea}
    },
    "features" : [
            <#list featureList?keys as key>
            {
                "type": "Feature",
                "id": "${featureList[key].name}",
                "geometry":
                {
                "type" : "Polygon",
                "coordinates" : ${featureList[key].geometryCoords}
            },
            "properties": {
                <#if featureList[key].owcProperties != "">$
↪{featureList[key].owcProperties},</#if>
                "offerings" : [
                    {
                      "code" : "http://www.opengis.net/spec/owc-atom/1.0/
↪req/wms",
                      "operations" : [{
                          "code" : "GetCapabilities",
                          "method" : "GET",
                          "type" : "application/xml",
                          "href" : "${featureList[key].getMapBaseUrl}?
↪SERVICE=WMS&VERSION=1.3.0&REQUEST=GetCapabilities",
                          "request":{},
                          "result":{}
                        },{
                          "code" : "GetMap",
                          "method" : "GET",
```

```
                        "type" : "image/png",
                        "href" : "${featureList[key].getMapBaseUrl}?
↪SERVICE=WMS&VERSION=1.1.1&REQUEST=GetMap&SRS=${featureList[key].srs}&
↪BBOX=${featureList[key].bbox}&WIDTH=500&HEIGHT=500&LAYERS=$
↪{featureList[key].layers}&STYLES=${featureList[key].styles}&
↪FORMAT=image/png&BGCOLOR=0xffffff&TRANSPARENT=TRUE&
↪EXCEPTIONS=application/vnd.ogc.se_xml",
                        "request":{},
                        "result":{}
                      }],
                    "contents" : []
                  }
              <#if featureList[key].type == "VECTOR">
                  ,{
                    "code" : "http://www.opengis.net/spec/owc-atom/1.0/
↪req/wfs",
                    "operations" : [{
                        "code" : "DescribeFeatureType",
                        "method" : "GET",
                        "type" : "application/xml",
                        "href" : "${featureList[key].getMapBaseUrl}?
↪SERVICE=WFS&VERSION=1.1.0&REQUEST=DescribeFeatureType&TYPENAME=$
↪{featureList[key].layers}",
                        "request":{},
                        "result":{}
                      },{
                        "code" : "GetFeature",
                        "method" : "GET",
                        "type" : "application/xml",
                        "href" : "${featureList[key].getMapBaseUrl}?
↪SERVICE=WFS&VERSION=1.1.0&REQUEST=GetFeature&TYPENAME=$
↪{featureList[key].layers}",
                        "request":{},
                        "result":{}
                      }],
                    "contents" : []
                  }
            <#elseif featureList[key].type == "RASTER">
                  ,{
                    "code" : "http://www.opengis.net/spec/owc-atom/1.0/
↪req/wcs",
                    "operations" : [{
                        "code" : "DescribeCoverage",
                        "method" : "GET",
                        "type" : "application/xml",
                        "href" : "${featureList[key].getMapBaseUrl}?
↪SERVICE=WCS&VERSION=1.1.0&REQUEST=GetCapabilities&IDENTIFIER=$
↪{featureList[key].layers}",
                        "request":{},
                        "result":{}
                      },{
                        "code" : "GetCoverage",
                        "method" : "GET",
                        "type" : "image/tiff",
                        "href" : "${featureList[key].getMapBaseUrl}?
↪SERVICE=WCS&VERSION=1.1.0&REQUEST=GetCoverage&IDENTIFIER=$
↪{featureList[key].layers}&BOUNDINGBOX=${featureList[key].bbox}&
↪FORMAT=GeoTIFF",
```

```
                          "request":{},
                          "result":{}
                        }],
                      "contents" : []
                    }
            </#if>
            ]
           }
         }<#if key_has_next>,</#if>
     </#list>
     ]
  ,

  "properties" : {
            ${owcProperties}
      }

}
```

Sample `owc_json_ctx.json`

```
"lang" : "en",
"title" : "Sample Title goes here",
"subtitle" : "Sample sub-title goes here",
"generator" : "Sample generator",
"rights" : "Sample Legal Constraints and CopyRights (C)",
"authors" : [{"name" : "Author1 Name"}, {"name" : "Author2 Name"}],
"contributors" : [{"name" : "Contrib1 Name"}, {"name" : "Contrib2 Name"}],
"categories" : [{
        "term" : "wms",
        "label" : "This file is compliant with version 1.0 of OGC Context"
    },{
        "term" : "maps",
        "label" : "This file contains maps"
}],
"links" : [{
        "rel" : "profile",
        "href" : "http://www.opengis.net/spec/owc-atom/1.0/req/core",
        "title" : "This file is compliant with version 1.0 of OGC Context"
    },{
        "rel" : "via",
        "type" : "application/xml",
        "href" : "http://www.opengis.uab.cat/wms/satcat/metadades/EPSG_
↪23031/Cat_20110301.htm",
        "title" : "HMTL metadata in Catalan"
    }]
```

Sample `result#.json`

```
"title" : "Result 2",
"updated" : "${lastUpdated}",
"content" : "Sample Content Description for result 2 goes here",
"authors" : [
    {
      "name" : "GeoServer Administrator",
      "email" : "info@sample.author.com"
    }
],
```

```
"authors" : [{"name" : "Author2.1 Name"}, {"name" : "Author2.2 Name"}],
"contributors" : [{"name" : "Contrib2.1 Name"}, {"name" : "Contrib2.2 Name
↪"}],
"categories" : [{"name" : "Category2.1 Name"}, {"name" : "Category2.2 Name
↪"}],
"links" : [
    {
     "rel" : "enclosure",
     "type" : "image/png",
     "title" : "WMS output for ${title}",
     "href" : "${getMapBaseUrl}?SERVICE=WMS&VERSION=1.1.1&REQUEST=GetMap&
↪SRS=${srs}&BBOX=${bbox}&WIDTH=500&HEIGHT=500&LAYERS=${layers}&
↪FORMAT=image/png&BGCOLOR=0xffffff&TRANSPARENT=TRUE&
↪EXCEPTIONS=application/vnd.ogc.se_xml"
    },
    {
     "rel" : "icon",
     "type" : "image/png",
     "title" : "Preview for ${title}",
     "href" : "${getMapBaseUrl}?SERVICE=WMS&VERSION=1.1.1&REQUEST=GetMap&
↪SRS=${srs}&BBOX=${bbox}&WIDTH=100&HEIGHT=100&LAYERS=${layers}&STYLES=$
↪{styles}&FORMAT=image/png&BGCOLOR=0xffffff&TRANSPARENT=TRUE&
↪EXCEPTIONS=application/vnd.ogc.se_xml"
    },
    {
     "rel" : "via",
     "type" : "application/vnd.ogc.wms_xml",
     "title" : "Original GetCapabilities document",
     "href" : "${getMapBaseUrl}?SERVICE=WMS&VERSION=1.1.1&
↪REQUEST=GetCapabilities"
    }
]
```

Other options for the Outputs

- **backup_on_wps_execution_shared_dir**; This is a boolean which tells to the Remote WPS to store first the outcome into the **sharedir** defined into the `[DEFAULT]` section before streaming out to GeoServer. This allows the Remote WPS to preserve the outcomes even when the resources are cleaned out.
- **upload_data**; This is a boolean which tells to the Remote WPS to upload first the outcome into the **host** defined into the `[UPLOADER]` section before streaming out to GeoServer. This allows the Remote WPS to preserve the outcomes even when the resources are cleaned out.

> **Warning:** If both enabled for a certain output, the **backup_on_wps_execution_shared_dir** takes precedence to the **upload_data** one.

- **publish_as_layer**; A boolean to instruct GeoServer Remote WPS to *try* to automatically publish the outcome as a new Layer through the GeoServer **Importer** Plugin.
- **publish_default_style**; The default style to use when publishing the Layer.
- **publish_target_workspace**; The default workspace to use when publishing the Layer.
- **publish_layer_name**; The default name to use when publishing the Layer.

Logging Section

```
# ######################################### #
# Logging RegEx and Levels                  #
# ######################################### #

[Logging]
stdout_parser = [.*\[DEBUG\](.*), .*\[INFO\] ProgressInfo\:([-+]?[0-9]*\.?[0-9]*)\%, .
↪*\[(INFO)\](.*), .*\[(WARN)\](.*), .*\[(ERROR)\](.*), .*\[(CRITICAL)\](.*)]
stdout_action = [ignore,          progress,                                          ␣
↪log,              log,              log,               abort]
```

- **stdout_parser**

  This property must contain a *list* of regular expressions matching the possible executable `STDOUT` logging messages the user wants to forward to GeoServer.

  As an instance

  ```
  .*\[DEBUG\](.*)
  ```

  Matches all the messages containing the keyword `[DEBUG]` and forwards to the corresponding **stdout_action** (*see below*) the content of the first matching group `(.*)`

  In this case everything after `[DEBUG]` is forwarded to the action.

  Another example

  ```
  .*\[INFO\] ProgressInfo\:([-+]?[0-9]*\.?[0-9]*)\%
  ```

  Matches all the messages containing the keyword `[INFO] ProgressInfo:<any_number>%` and forwards to the corresponding **stdout_action** (*see below*) the content of the first matching group `([-+]?[0-9]*\.?[0-9]*)`

  In this case the expression extracts a float number form the text along with the sign `[-+]`

- **stdout_action**

  This property must contain a *list* of keywords associated to a *particular action* which will take the content of the corresponding regular expression and forwards it to GeoServer packaged ad a specific XMPP message.

  As an instance

  - *progress*; gets the content of the match and sends a **PROGRESS** XMPP message to GeoServer. The **PROGRESS** messgae must always contain a number.
  - *abort*; gets the content of the match and sends a **ABORT** XMPP message to GeoServer. This will cause GeoServer to mark the WPS Process as **FAILED**.
  - *ignore*; simply throws out everything matching the corresponding regular expression.
  - *log*; sends a **LOG** message to GeoServer with the content of the match. This will appear into the GeoServer Log file.

A Remote "Gdal Contour" Process Binding Example

Before continue reading this section, please be sure to have fully understood and successfully completed all the passages at sections:

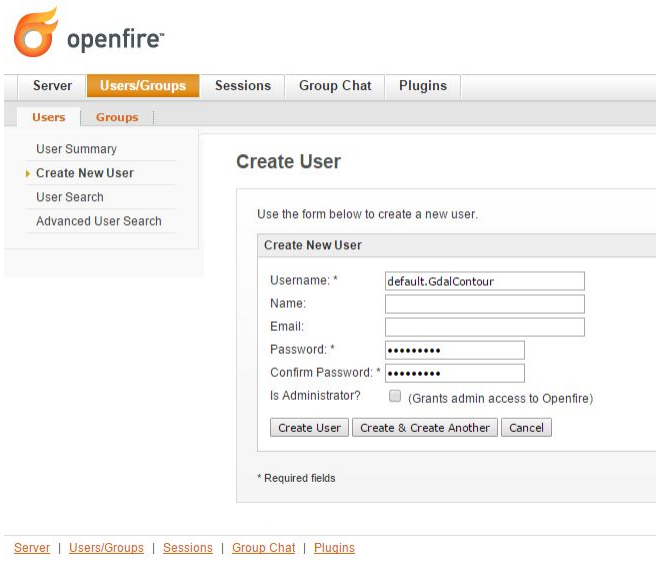
- *Deployment And Setup Of GeoServer With WPS Remote Plugin*
- *Installation Of OpenFire XMPP Server To Exchange Messages*
- *Deployment And Setup Of The XMPP Python Wrappers*

Running the Python WPS Agent

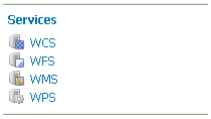
In order to start the RemoteWPS Python Wrapper, we need to run an instance of the `wpsagent.py` using the configuration files defined at section *Deployment And Setup Of The XMPP Python Wrappers*

```
$> cd C:\work\RemoteWPS

$> python wpsagent.py -r .\xmpp_data\configs\remote.config -s .\xmpp_
↪data\configs\myservice\service.config service
```

Few instants after the execution of the command, you should be able to see con `invite` message on the prompt



and the `default.GdalContour` instance successfully connected and authenticated into the XMPP Server channels

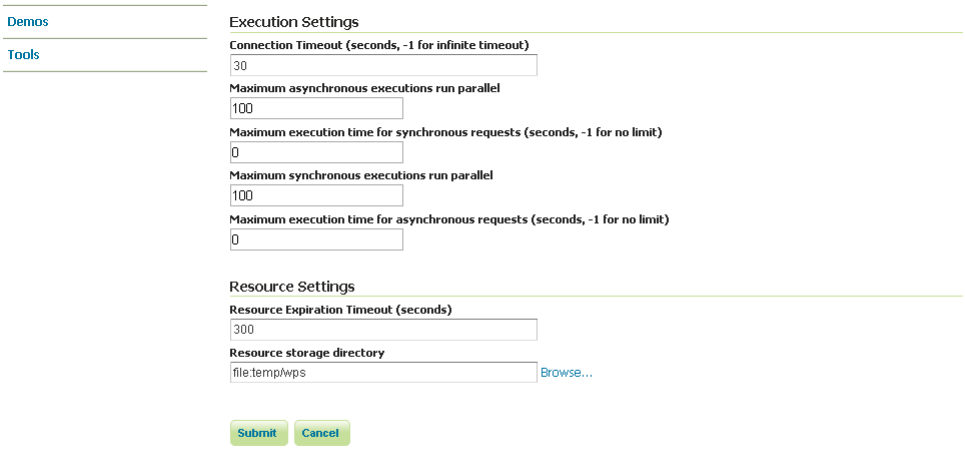


The new GeoServer WPS Process should be now available among the GeoServer Processes

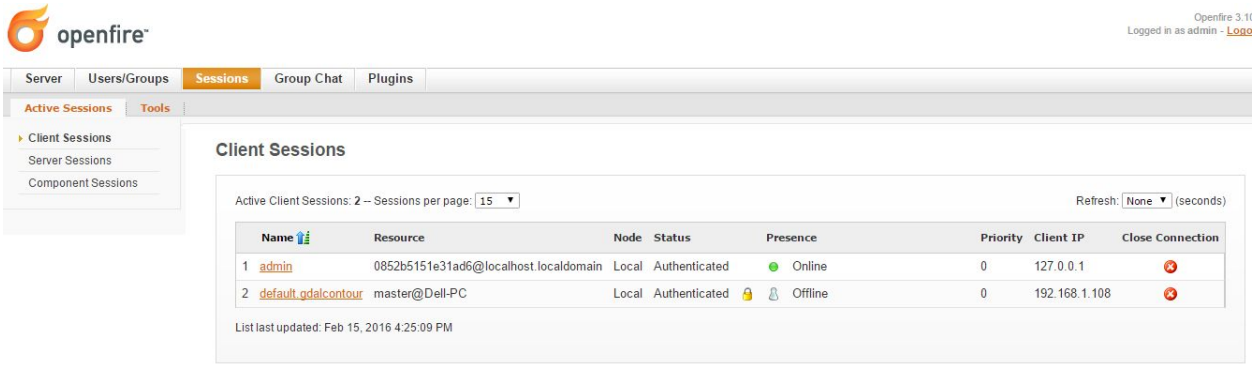
The GeoServer Remote Process Factory automatically creates the WPS interface for the new process, exposing through the OGC WPS Protocol the Inputs and Outputs definitions like shown in the illustration below

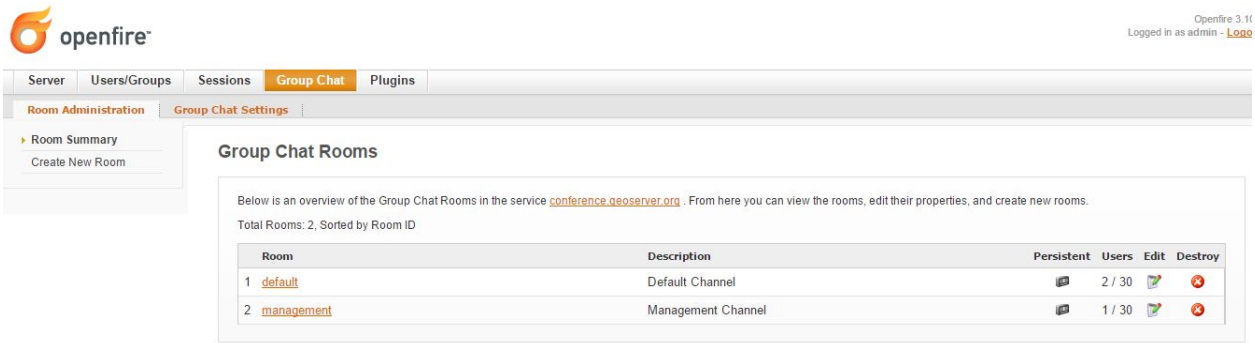
At the Execute Request the Remote WPS Python framework starts a new thread and assigns to it the unique **execution_id** provided by GeoServer.

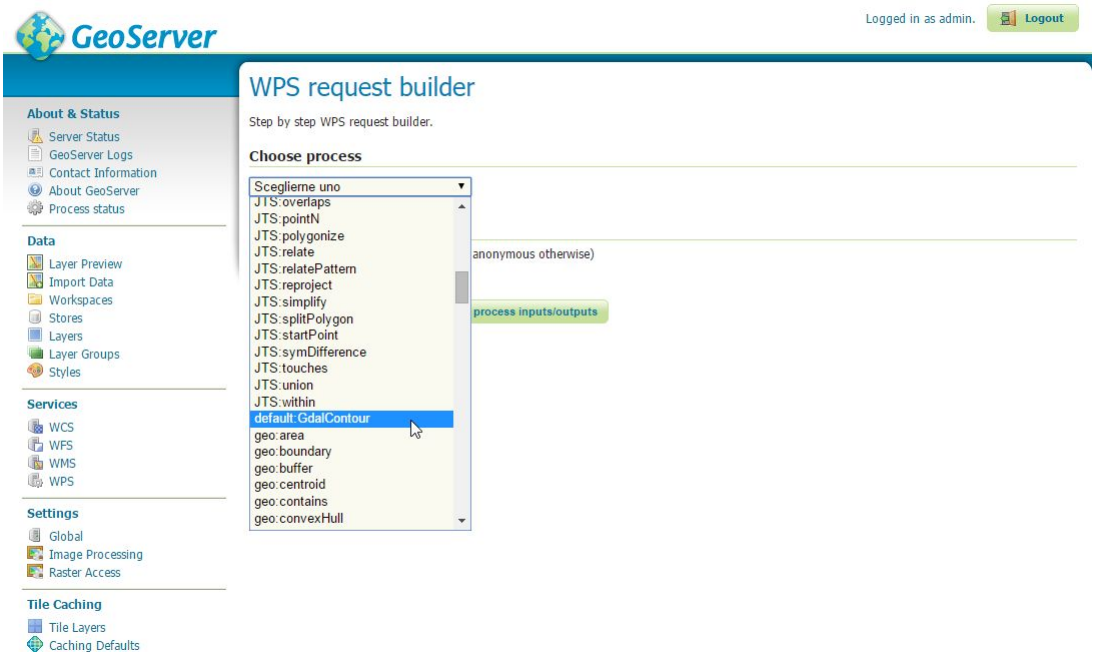
The logs of the execution are stored into the **working directory**




```
C:\tmp\RemoteWPS>python wpsagent.py -r .\xmpp_data\configs\remote.config -s .\xmpp_data\configs\myservice\service.config service
2016-02-15 16:00:50,607 [DEBUG] Logger initialized with file .\xmpp_data\configs\logger.properties
2016-02-15 16:00:50,608 [INFO] Create process bot
2016-02-15 16:00:52,788 [INFO] Create resource cleaner
2016-02-15 16:00:52,789 [INFO] Start bot execution
2016-02-15 16:00:52,789 [INFO] start resource cleaner main loop
2016-02-15 16:00:52,789 [INFO] Start listening on bus
2016-02-15 16:00:52,789 [DEBUG] Sleep for 300.0 seconds
2016-02-15 16:00:53,424 [INFO] Received XMPP bus signal from default@conference.geoserver.org/admin@geoserver.org: "topic=invite"
2016-02-15 16:00:53,453 [INFO] handle invite message from WPS default@conference.geoserver.org/admin@geoserver.org
2016-02-15 16:01:11,795 [INFO] Received XMPP bus signal from default@conference.geoserver.org/admin@geoserver.org: "topic=request&id=3fa7cf1fdf382af&baseURL=http://192
.168.1.122:8080/geoserver/&message=%80%02%7Dq%00%28X%08%00%00%00intervalq%01Kdu."
2016-02-15 16:01:11,825 [DEBUG] save parameters file for executing process GdalContour in c:\users\dell\appdata\local\temp\wps_params_ydrhjh.tmp
2016-02-15 16:01:11,834 [INFO] created process GdalContour with PId 1240 and cmd: python wpsagent.py -r .\xmpp_data\configs\remote.config -s .\xmpp_data\configs\myserv
ice\service.config -p c:\users\dell\appdata\local\temp\wps_params_ydrhjh.tmp process
2016-02-15 16:01:11,835 [INFO] end of execute message handler, going back in listening mode
2016-02-15 16:01:11,835 [INFO] wait for end of execution of created process GdalContour, PId 1240
```







From the log file is possible to recognize the full command line executed by the Remote WPS Python wrapper along with the lines received through the standard output

The main window shows the received XMPP messages and the actions taken accordingly

---

Note: The same information can be found into the log file specified into the "logger.properties" file (see above).

On GeoServer side, it is possible to follow the process execution by following the messages sent via XMPP to the GeoServer logs

```
$> tail -F -n 200 /storage/data/logs/geoserver.log
```

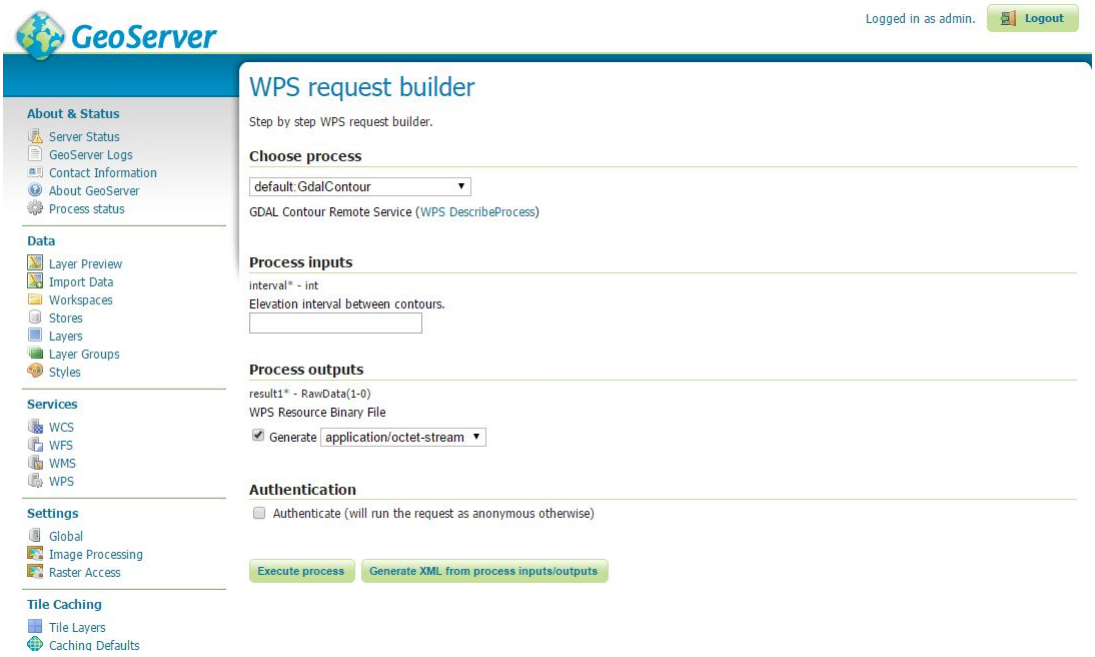
16.22 JDBCStore

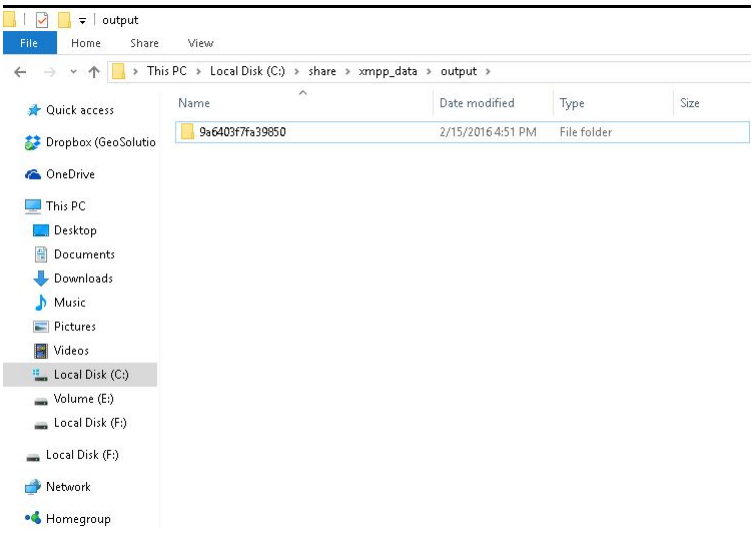
The `JDBCStore module` allows efficient sharing of configuration data in a clustered deployment of GeoServer. It allows externalising the storage of all configuration resources to a Relational Database Management System, rather than using the default File System based *GeoServer data directory*. This way the multiple instances of GeoServer can use the same Database and therefore share in the same configuration.

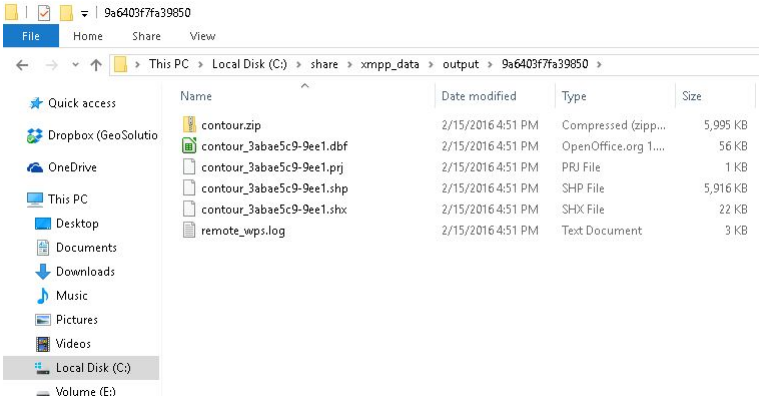
16.22.1 Installing JDBCStore

To install the JDBCStore module:

1. Download the module. The file name is called `geoserver-*-jdbcstore-plugin.zip`, where `*` is the version/snapshot name. The JDBCStore plug-in automatically includes the *JDBCConfig* plugin as well which will generally be run at the same time.
2. Extract this file and place the JARs in `WEB-INF/lib`.


GeoServer
Logged in as admin.
Logout
About & Status
Server Status
GeoServer Logs
Contact Information
About GeoServer
Process status
Data
Layer Preview
Import Data
Workspaces
Stores
Layers
Layer Groups
Styles
Services
WCS
WFS
WMS
WPS
Settings
Global
Image Processing
Raster Access
Tile Caching
Tile Layers
Caching Defaults
WPS request builder
Step by step WPS request builder.
Choose process
default:GdalContour
GDAL Contour Remote Service (WPS DescribeProcess)
Process inputs
interval* - int
Elevation interval between contours.
Process outputs
result1* - RawData(1-0)
WPS Resource Binary File
Generate
application/octet-stream
Authentication
Authenticate (will run the request as anonymous otherwise)
Execute process
Generate XML from process inputs/outputs


output
File
Home
Share
View
This PC
Local Disk (C:)
share
xmpp_data
output
Quick access
Dropbox (GeoSolutio
OneDrive
This PC
Desktop
Documents
Downloads
Music
Pictures
Videos
Local Disk (C:)
Volume (E:)
Local Disk (F:)
Local Disk (F:)
Network
Homegroup
Name
Date modified
Type
Size
9a6403f7fa39850
2/15/2016 4:51 PM
File folder


9a6403f7fa39850
File
Home
Share
View
This PC
Local Disk (C:)
share
xmpp_data
output
9a6403f7fa39850
Quick access
Dropbox (GeoSolutio
OneDrive
This PC
Desktop
Documents
Downloads
Music
Pictures
Videos
Local Disk (C:)
Volume (E:)
Name
Date modified
Type
Size
contour.zip
2/15/2016 4:51 PM
Compressed (zipp...
5,995 KB
contour_3abae5c9-9ee1.dbf
2/15/2016 4:51 PM
OpenOffice.org 1....
56 KB
contour_3abae5c9-9ee1.prj
2/15/2016 4:51 PM
PRJ File
1 KB
contour_3abae5c9-9ee1.shp
2/15/2016 4:51 PM
SHP File
5,916 KB
contour_3abae5c9-9ee1.shx
2/15/2016 4:51 PM
SHX File
22 KB
remote_wps.log
2/15/2016 4:51 PM
Text Document
3 KB

3. Perform any configuration required by your servlet container, and then restart. On startup, JDBCStore will create a configuration directory `jdbcstore` and JDBCConfig will create a configuration directory `jdbcconfig` in the *GeoServer data directory* .
4. Verify that the configuration directories were created to be sure installation worked then turn off GeoServer.
5. If you want to use *JDBCConfig* as well, configure it first, being sure to set `enabled`, `initdb`, and `import` to `true`, and to provide the connection information for an empty database. Start GeoServer to initialize the JDBCConfig database, import the old catalog into it, and take over from the old catalog. Subsequent start ups will skip the initialize and import steps unless you re-enable them in `jdbcconfig.properties`.
6. Now configure JDBCStore in a similar fashion (*JDBCStore configuration*), being sure to set `enabled`, `initdb`, and `import` to `true`, and to provide the connection information for an empty database. Start GeoServer again. This time JDBCStore will connect to the specified database, initialize it, import the old *GeoServer data directory* into it, and take over from the old *GeoServer data directory*. Subsequent start ups will skip the initialize and import steps unless you re-enable them in `jdbcstore.properties`.

16.22.2 JDBCStore configuration

The JDBCStore module is configured in the file `jdbcstore/jdbcstore.properties` inside the *GeoServer data directory*. The following properties may be set:

- `enabled`: Use JDBCStore. Turn off to use the data directory for all configuration instead.
- `initdb`: Initialize an empty database if this is set on true.
- `import` : The import configuration option tells GeoServer whether to import the current *GeoServer data directory* from the file system to the database or not. If set to true, it will be imported and the config option will be set the value 'false' for the next start up to avoid trying to re-import the catalog configuration.
- `initScript`: Path to initialisation script .sql file. Only used if `initdb` is true.
- `ignoreDirs`: specify all subdirectories of the *GeoServer data directory* that should be ignored by the JDBCStore, in a comma-separate list. These subdirectories will not be imported and while JDBCStore is running, all access to these subdirectories and their contents will be redirected to the default file system store. This is usually done with the `data` directory (which holds data rather than metadata such as images and shapefiles), temporary directories (which are not used for permanent storage) and the catalog directories (when using JDBCConfig, these are unused anyway and they need not be copied into the JDBCStore).
- `deleteDestinationOnRename`: allow automatic overwriting of existing destinations on move and rename operations (linux-style versus windows-style - the default store is platform dependant).

JNDI

Get the database connection from the application server via JNDI lookup.

- `jndiName`: The JNDI name for the data source. Only set this if you want to use JNDI, the JDBC configuration properties may still be set for in case the JNDI Lookup fails.

Direct JDBC Connection

Provide the connection parameters directly in the configuration file. This includes the password in the clear which is a potential security risk. To avoid this use JNDI instead.

- `jdbcUrl`: JDBC direct connection parameters.
- `username`: JDBC connection username.
- `password`: JDBC connection password.
- `pool.minIdle`: minimum connections in pool
- `pool.maxActive`: maximum connections in pool
- `pool.poolPreparedStatements`: whether to pool prepared statements
- `pool.maxOpenPreparedStatements`: size of prepared statement cache, only used if `pool.poolPreparedStatements` is true
- `pool.testOnBorrow`: whether to validate connections when obtaining from the pool
- `pool.validationQuery`: validation query for connections from pool, must be set when `pool.testOnBorrow` is true

16.23 ncWMS WMS extensions support

The **ncWMS module** adds to GeoServer the ability to support some of the ncWMS extensions to the WMS protocol and configuration. In particular:

- Ability to create a named style by simply providing a list of colors, that will adapt to the layer in use based on request parameters and its statistics
- Ability to control the palette application in GetMap via a number of extra parameters
- GetTimeSeries operation, which can retrieve a CSV or an xy chart of a time series of values on a certain point

At the time of writing the extra calls to extract elevation series, transects and NetCDF metadata are not supported. The extension is however not NetCDF specific, but can be used with any single banded raster layer instead.

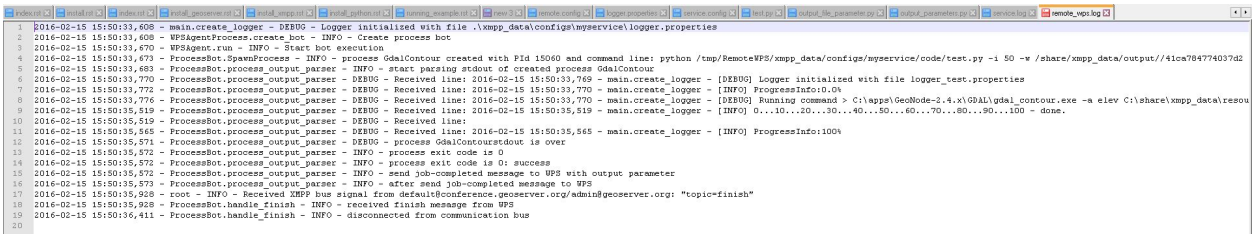
16.23.1 The Dynamic Palette style format

A new "Dynamic palette" style format has been added that accepts a palette, one color per line, defining a color progression to be applied on raster data. Each color can be defined using these possible syntaxes (same as ncWMS):

- `#RRGGBB`
- `#AARRGGBB`
- `0xRRGGBB`
- `0xAARRGGBB`

Comments can be added in the file by starting the line by a percentage sign. For example, a red to blue progression might look like:

```
% Red to blue progression
#FF0000
#0000FF
```



Several ready to use palettes coming from the popular "color brewer" site are available in the ncWMS source code repository.

The palette translates on the fly into a SLD with rendering transformation using the *Dynamic colormap generation* module, in particular, the above style translates to the following style:

```
<?xml version="1.0" encoding="UTF-8"?>
<sld:StyledLayerDescriptor xmlns="http://www.opengis.net/sld" xmlns:sld="http://www.
↪opengis.net/sld" xmlns:gml="http://www.opengis.net/gml" xmlns:ogc="http://www.
↪opengis.net/ogc" version="1.0.0">
  <sld:NamedLayer>
    <sld:Name/>
    <sld:UserStyle>
      <sld:Name/>
      <sld:FeatureTypeStyle>
        <sld:Transformation>
          <ogc:Function name="ras:DynamicColorMap">
            <ogc:Function name="parameter">
              <ogc:Literal>data</ogc:Literal>
            </ogc:Function>
            <ogc:Function name="parameter">
              <ogc:Literal>opacity</ogc:Literal>
              <ogc:Function name="env">
                <ogc:Literal>OPACITY</ogc:Literal>
                <ogc:Literal>1.0</ogc:Literal>
              </ogc:Function>
            </ogc:Function>
            <ogc:Function name="parameter">
              <ogc:Literal>colorRamp</ogc:Literal>
              <ogc:Function name="colormap">
                <ogc:Literal>rgb(255,0,0);rgb(0,0,255)</ogc:Literal>
                <ogc:Function name="env">
                  <ogc:Literal>COLORSCALERANGE_MIN</ogc:Literal>
                  <ogc:Function name="bandStats">
                    <ogc:Literal>0</ogc:Literal>
                    <ogc:Literal>minimum</ogc:Literal>
                  </ogc:Function>
                </ogc:Function>
                <ogc:Function name="env">
                  <ogc:Literal>COLORSCALERANGE_MAX</ogc:Literal>
                  <ogc:Function name="bandStats">
                    <ogc:Literal>0</ogc:Literal>
                    <ogc:Literal>maximum</ogc:Literal>
                  </ogc:Function>
                </ogc:Function>
```

```
                <ogc:Function name="env">
                  <ogc:Literal>BELOWMINCOLOR</ogc:Literal>
                  <ogc:Literal>rgba(0,0,0,0)</ogc:Literal>
                </ogc:Function>
                <ogc:Function name="env">
                  <ogc:Literal>ABOVEMAXCOLOR</ogc:Literal>
                  <ogc:Literal>rgba(0,0,0,0)</ogc:Literal>
                </ogc:Function>
                <ogc:Function name="env">
                  <ogc:Literal>LOGSCALE</ogc:Literal>
                  <ogc:Literal>false</ogc:Literal>
                </ogc:Function>
                <ogc:Function name="env">
                  <ogc:Literal>NUMCOLORBANDS</ogc:Literal>
                  <ogc:Literal>254</ogc:Literal>
                </ogc:Function>
              </ogc:Function>
            </ogc:Function>
          </ogc:Function>
        </sld:Transformation>
        <sld:Rule>
          <sld:RasterSymbolizer/>
        </sld:Rule>
      </sld:FeatureTypeStyle>
    </sld:UserStyle>
  </sld:NamedLayer>
</sld:StyledLayerDescriptor>
```

The above explains a bit of how the palette is applied:

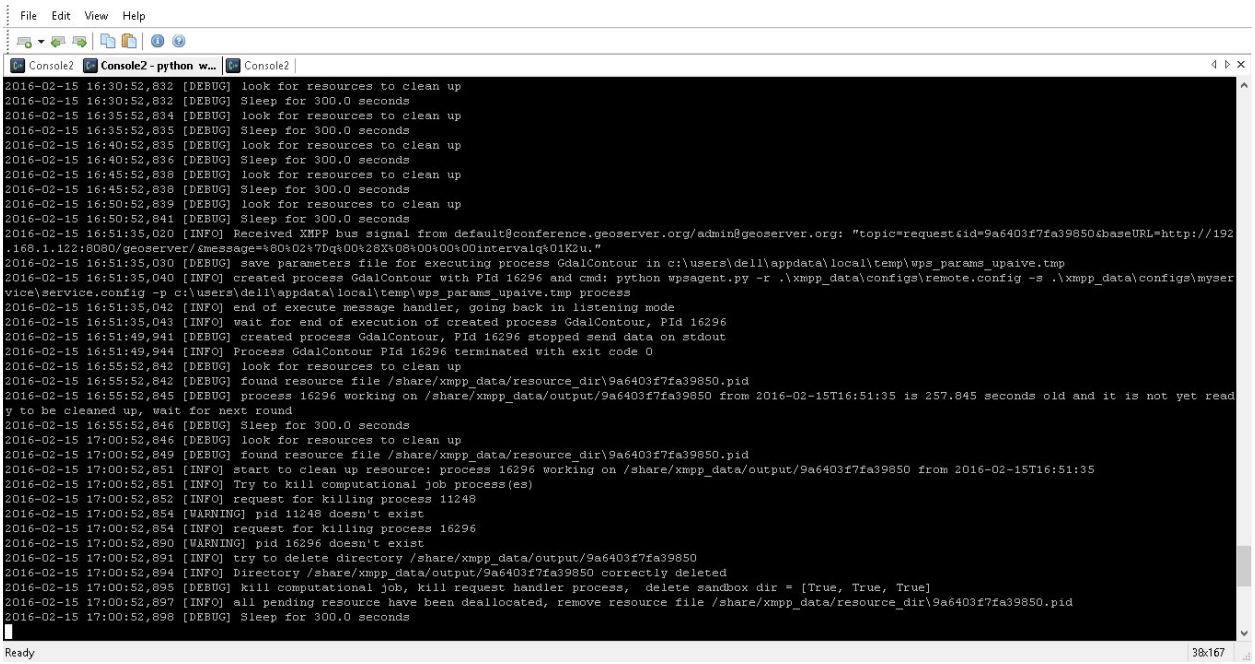
- By default a palette of 254 colors is generated between a min and max value, plus one color for anything below the minimum and another for anything above the maximum
- It is possible to pass the minimum and maximum values using the GetMap `env` parameter, if not provided, they are fetched from the configured band statistics (as found in the layer configuration)
- The overall opacity of the palette can be controlled (using a value between 0 and 1 to conform with the SLD opacity description)
- The scale can be either linear, or logarithmic

The above parameters can all be used at will to control the palette generation using the typical environment variable approach. However, it's also possible to use ncWMS own extensions, which are adding direct parameters in the request. See the following section for details.

16.23.2 ncWMS GetMap extensions

This module also adds a dynamic translator taking the ncWMS GetMap vendor parameters and mapping them to the dynamic palette expectations. In particular (copying the parameter description from the ncWMS manual, with GeoServer specific annotations):

- COLORSCALERANGE: Of the form min,max this is the scale range used for plotting the data (mapped to the COLORSCALERANGE_MIN and COLORSCALERANGE_MAX env vars)
- NUMCOLORBANDS: The number of discrete colours to plot the data. Must be between 2 and 250 (mapped to the NUMCOLORBANDS env variable)
- ABOVEMAXCOLOR: The colour to plot values which are above the maximum end of the scale range. Colours are of the form 0xRRGGBB or 0xAARRGGBB, and it also accepts "transparent" and "extend"



- BELOWMINCOLOR: The colour to plot values which are below the minimum end of the scale range. Colours are of the form 0xRRGGBB or 0xAARRGGBB, and it also accepts "transparent" and "extend"
- LOGSCALE: "true" or "false" - whether to plot data with a logarithmic scale
- OPACITY: The percentage opacity of the final output image as a number between 0 and 100 (maps to OPACITY env var by translating it to a number between 0 and 1)
- ANIMATION: "true" or "false" - whether to generate an animation. The ncWMS documentation states that TIME has to be of the form `starttime/endtime`, but currently TIME needs to be a list of discrete times instead. Animation requires using the "image/gif" as the response format (as the only format supporting animation)

Here are a few examples based on the "ArcSample" arcgrid sample layer, containing annual precipitation data. The one band provided by this layer has been configured with a default range of 0 to 6000.

- Default output with the "redblue" palette:

  http://localhost:8080/geoserver/wms?STYLES=redblue&LAYERS=nurc%3AArc_Sample&FORMAT=image%2Fpng&SERVICE=WMS&VERSION=1.1.1&REQUEST=GetMap&SRS=EPSG%3A4326&BBOX=-180,-90,180,90&WIDTH=500&HEIGHT=250

- Adopting a logarithmic scale by adding `&COLORSCALERANGE=1,6000&LOGSCALE=true` (a logarithmic scale needs a positive minimum)
- Using just 5 colors in logarithmic mode by adding `&COLORSCALERANGE=1,6000&LOGSCALE=true&NUMCOLORBANDS=5`
- Limiting the range and specifying other colors above (gray) and below (yellow) by adding `&COLORSCALERANGE=100,2000&BELOWMINCOLOR=0xFFFF00&ABOVEMAXCOLOR=0xAAAAAA`

16.23.3 ncWMS GetCapabilities extensions

ncWMS allows users to filter the contents of a capabilites document by adding a `&dataset=datasetName` parameter to the request.

```
          mimeType = application/octet-stream
2016-02-15 16:51:38,186 INFO [geoserver.wps] -
Request: getServiceInfo
2016-02-15 16:51:38,345 INFO [wps.remote] - Generating a unique Process ID for Remote Process [default:GdalContour] with the following parameters:
2016-02-15 16:51:38,346 INFO [wps.remote] -  - name: default:GdalContour
2016-02-15 16:51:38,350 INFO [wps.remote] -  - input: {interval=50}
2016-02-15 16:51:38,350 INFO [wps.remote] -  - metadata: {serviceJID=default@conference.geoserver.org/GdalContour@geoserver.org/master@Dell-PC, requ
est=WPS 1.0.0 Execute}
2016-02-15 16:51:38,350 INFO [wps.remote] -  - monitor: org.geoserver.wps.executor.ProcessStartupFilter$ProcessStartupWrapper$1@44fd9ca2
2016-02-15 16:51:38,350 INFO [remote.plugin] - XMPPClient::getFlattestMachine - scanning the connected remote services...
2016-02-15 16:51:38,350 INFO [remote.plugin] - XMPPClient::getFlattestMachine - looking for service [GdalContour] @occupant [default@conference.geos
erver.org/admin@geoserver.org]
2016-02-15 16:51:38,350 INFO [remote.plugin] - XMPPClient::getFlattestMachine - looking for service [GdalContour] @occupant [default@conference.geos
erver.org/GdalContour@geoserver.org/master@Dell-PC]
2016-02-15 16:51:38,350 INFO [remote.plugin] - XMPPClient::getFlattestMachine - [localizedServiceJID.length] -> 2
2016-02-15 16:51:38,350 INFO [remote.plugin] - XMPPClient::getFlattestMachine - [localizedServiceJID[0].contains(serviceName)] -> true
2016-02-15 16:51:38,350 INFO [remote.plugin] - XMPPClient::getFlattestMachine - [localizedServiceJID] -> GdalContour @ geoserver.org
2016-02-15 16:51:38,350 INFO [remote.plugin] - XMPPClient::getFlattestMachine - target JID found, using the target [default@conference.geoserver.org
/GdalContour@geoserver.org/master@Dell-PC]
2016-02-15 16:51:38,350 INFO [remote.plugin] - XMPPClient::execute - trying to send a request message to the service JID [default@conference.geoserv
er.org/GdalContour@geoserver.org/master@Dell-PC]
2016-02-15 16:51:38,350 INFO [remote.plugin] - XMPPClient::execute - extracting the PID for the service JID [default@conference.geoserver.org/GdalCo
ntour@geoserver.org/master@Dell-PC] with inputs [{interval=50}]
2016-02-15 16:51:38,351 INFO [wps.remote] - Starting the execution of Remote Process with pId [9a6403f7fa39850]
2016-02-15 16:51:51,124 INFO [remote.plugin] - [9a6403f7fa39850]0...10...20...30...40...50...60...70...80...90...100 - done.
2016-02-15 16:51:51,195 INFO [plugin.output] -  - TEST - [XMPP Raw Data Output - ProduceOutput] /share/xmpp_data/output//contour.zip - type:applicat
ion/zip - pID:9a6403f7fa39850 - name:contour - title:SRTM - description:WPS Resource Binary File - style:polygon - workspace:it.geosolutions - metad
ata:
2016-02-15 16:51:51,195 INFO [plugin.output] -  - TEST - [XMPP Raw Data Output - ProduceOutput] FileRawData:contour_9a6403f7fa39850.zip
2016-02-15 16:51:51,212 INFO [plugin.output] -  - TEST - [XMPP Raw Data Output - ProduceOutput] Value:FileRawData [file=ResourceAdaptor(/share/xmpp_
data/output/contour.zip), mimeType=application/zip, extension=zip]
2016-02-15 16:51:51,212 INFO [remote.plugin] -  - TEST - [XMPPCompletedMessage] wpsOutputValue:FileRawData [file=ResourceAdaptor(/share/xmpp_data/ou
tput/contour.zip), mimeType=application/zip, extension=zip]
2016-02-15 16:51:51,223 INFO [wps.remote] - Stopping the execution of Remote Process with pId [9a6403f7fa39850]
2016-02-15 16:51:51,231 INFO [geoserver.wps] -
Request: execute
    service = WPS
    version = 1.0.0
    baseUrl = http://192.168.1.122:8080/geoserver/
    identifier:
        value = default:GdalContour
    dataInputs:
        input[0]:
            identifier = net.opengis.ows11.impl.CodeTypeImpl@5750fdfd (value: interval, codeSpace: null)
            data = net.opengis.wps10.impl.DataTypeImpl@4ca7752a
    responseForm:
        rawDataOutput:
            identifier = net.opengis.ows11.impl.CodeTypeImpl@567923dd (value: result1, codeSpace: null)
            mimeType = application/octet-stream
```

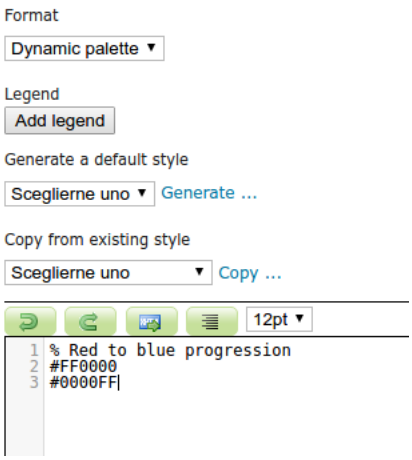


Fig. 16.18: *Configuring a dynamic palette style*

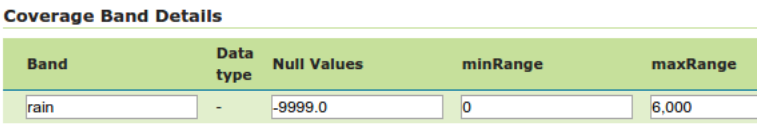
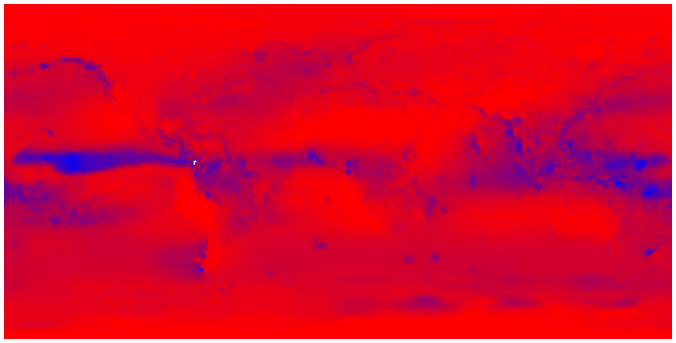


Fig. 16.19: *Editing the defaults for min/max scale range values in the GeoServer layer editor*



While GeoServer does not have a concept of dataset, the ncWMS extension allows to use the same parameter to filter on workspaces, layers and layer groups, by name.

For example:

- Getting everything in the "topp" workspace: http://localhost:8080/geoserver/ows?service=wms&version=1.3.0&request=GetCapabilities&dataset=topp
- Getting only the "topp:states" layer: http://localhost:8080/geoserver/ows?service=wms&version=1.3.0&request=Ge
- Getting the "tasmania" layer group: http://localhost:8080/geoserver/ows?service=wms&version=1.3.0&request=GetCapabilities&dataset=tasmania

16.23.4 ncWMS GetTimeSeries operation

ncWMS provides a GetTimeSeries operation, which can retrieve a time series of values on a certain point, using a syntax similar to the GetFeatureInfo operation. The time series can be retrieved as a chart in PNG or JPEG image, or as a CSV.

For example:

- Getting a time series as CSV: http://localhost:8080/geoserver/wms?SERVICE=WMS&VERSION=1.1.1&REQUEST=GetTimeSeries&FORMAT=image%2Fjpeg&TIME=2008-10-31T00:00:00.000Z/2008-11-01T00:00:00.000Z&QUERY_LAYERS=watertemp&STYLES&LAYERS=watertemp&INFO_FORMAT=text%2Fcsv&FEATURE_COUNT=50&X=50&Y=50&SRS=EPSG%3A4326&WIDTH=101&HEIGHT=101&BBOX=3.724365234375%2C40.81420898437501%2C5.943603515625%2C43.03344726562501
- Getting a time series as PNG: http://localhost:8080/geoserver/wms?SERVICE=WMS&VERSION=1.1.1&REQUEST=GetTimeSeries&FORMAT=image%2Fjpeg&TIME=2008-10-31T00:00:00.000Z/2008-11-01T00:00:00.000Z&QUERY_LAYERS=watertemp&STYLES&LAYERS=watertemp&INFO_FORMAT=image%2Fpng&FEATURE_COUNT=50&X=50&Y=50&SRS=EPSG%3A4326&WIDTH=101&HEIGHT=101&BBOX=3.724365234375%2C40.81420898437501%2C5.943603515625%2C43.03344726562501
- Getting a time series as JPG: http://localhost:8080/geoserver/wms?SERVICE=WMS&VERSION=1.1.1&REQUEST=GetTimeSeries&FORMAT=image%2Fjpeg&TIME=2008-10-31T00:00:00.000Z/2008-11-01T00:00:00.000Z&QUERY_LAYERS=watertemp&STYLES&LAYERS=watertemp&INFO_FORMAT=image%2Fjpg&FEATURE_COUNT=50&X=50&Y=50&SRS=EPSG%3A4326&WIDTH=101&HEIGHT=101&BBOX=3.724365234375%2C40.81420898437501%2C5.943603515625%2C43.03344726562501

The INFO_FORMAT accepts the following values: *image/png*, *image/jpg* and *text/csv*

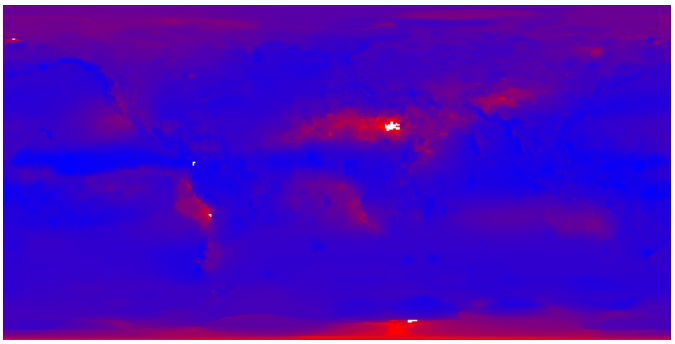
The TIME parameter accepts a time range as defined for other operations in the WMS standard (see Annex D of the 06-042 Web Map Server Implementation Specification). Examples:

- *TIME=2008-10-31T00:00:00.000Z/2008-11-01T00:00:00.000Z*
- *TIME=2008-10-31T00:00:00.000Z/2008-10-31T00:00:00.000Z*

Sample CSV output:

```
# Latitude: 40.396728515625
# Longitude: -0.6921386718750019
Time (UTC),Temperature (degrees)
2014-01-01T00:00:00.000Z,0.4194810092449188
2014-02-01T00:00:00.000Z,0.8373379707336426
2014-03-01T00:00:00.000Z,3.1670899391174316
2014-04-01T00:00:00.000Z,4.932330131530762
```

Sample chart output:



16.24 Backup and Restore Documentation

This section describes the GeoServer Backup and Restore plugin functionality and APIs.

16.24.1 Installation

Manual Install

To download and install the required extensions by hand:

1. Download the geoserver-2.15-SNAPSHOT-backup-restore-plugin.zip from:
   - Community Builds (GeoServer WebSite)

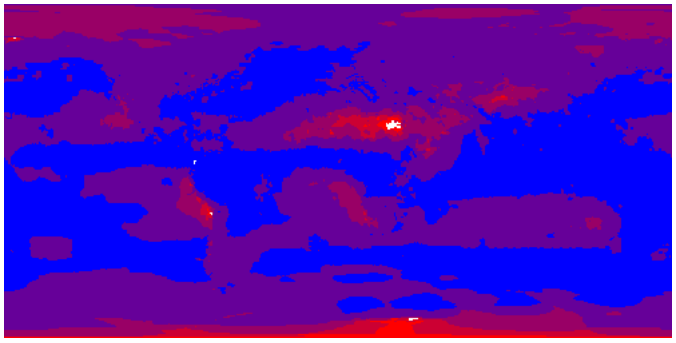
   It is important to download the version that matches the GeoServer you are running.
2. Stop the GeoServer application.
3. Navigate into the `webapps/geoserver/WEB-INF/lib` folder.

   These files make up the running GeoServer application.
4. Unzip the contents of the three zip files into the `lib` folder.
5. Restart the Application Server.
6. Login to the Web Administration application. Select **Data** from the naviagion menu. Click *Backup and Restore* and ensure the page is rendered correctly and without errors.

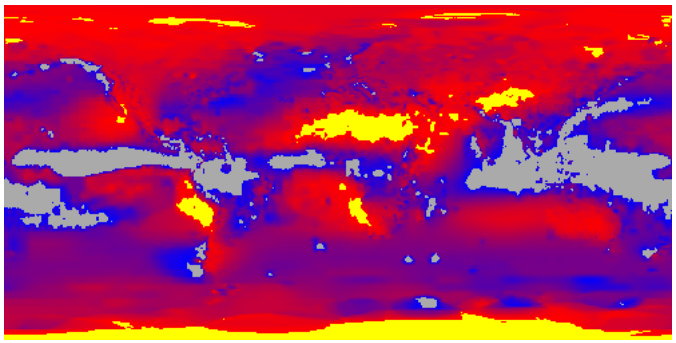
Backup and Restore plugin can be used both via user interface and via HTTP REST interface. For more details please see the next sections.

16.24.2 Usage Via GeoServer's User Interface

At the end on Backup and Restore plugin installation you will see a new section in GeoServer UI



Clicking on the `Backup and Restore` label will give you access to the Backup and Restore configuration settings:



Here you'll be able to specify various parameters for the Backup / Restore procedure:

1. `Archive full path`: Path on the file system to the archive created by the backup procedure, in case a Backup is executed, or the archive to restore from, in case of a Restore procedure.
2. `Filter by Workspace`: Optional parameter that allows you to restrict the scope of the Backup / Restore to workspaces that meet the specified filter.
3. `Backup Options`:
   (a) `Overwrite Existing Archive`: When enabled the backup procedure will overwrite any previously existing archive
   (b) `Skip Failing Resources`: If enabled and errors are found during the backup of existing resources, skip the resource and go ahead with the backup procedure
4. `Backup Executions`: Report of running and previously run backups
5. `Restore Options`:
   (a) `Dry Run`: Test the restore procedure using the provided archive but do not apply any changes to current configuration. Useful to test archives before actually performing a Restore

(b) `Skip Failing Resources`: If enabled and errors are found during the restore of resources, skip the resource and go ahead with the restore procedure

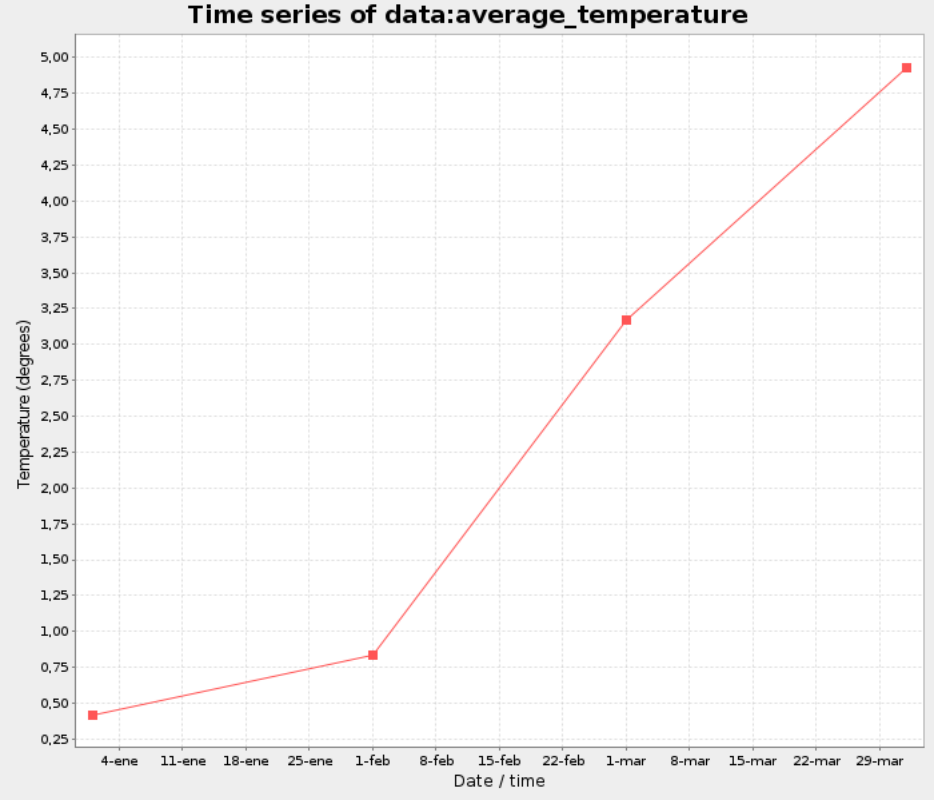
6. `Restore Executions`: Report of running and previously run restore

Performing a full backup via UI

In order to perform a full backup, provide the full path of the target `.zip` archive where to store the configuration data.

Note: Please notice that the backup will store just the configuration files and **not** the original data.

It is also possible to use the `Browse` instrument to navigate the server folders. In any case the backup procedure won't start until it find a valid `.zip` path archive.



It is possible to select the backup options by enabling the appropriate checkboxes before starting the backup procedure.

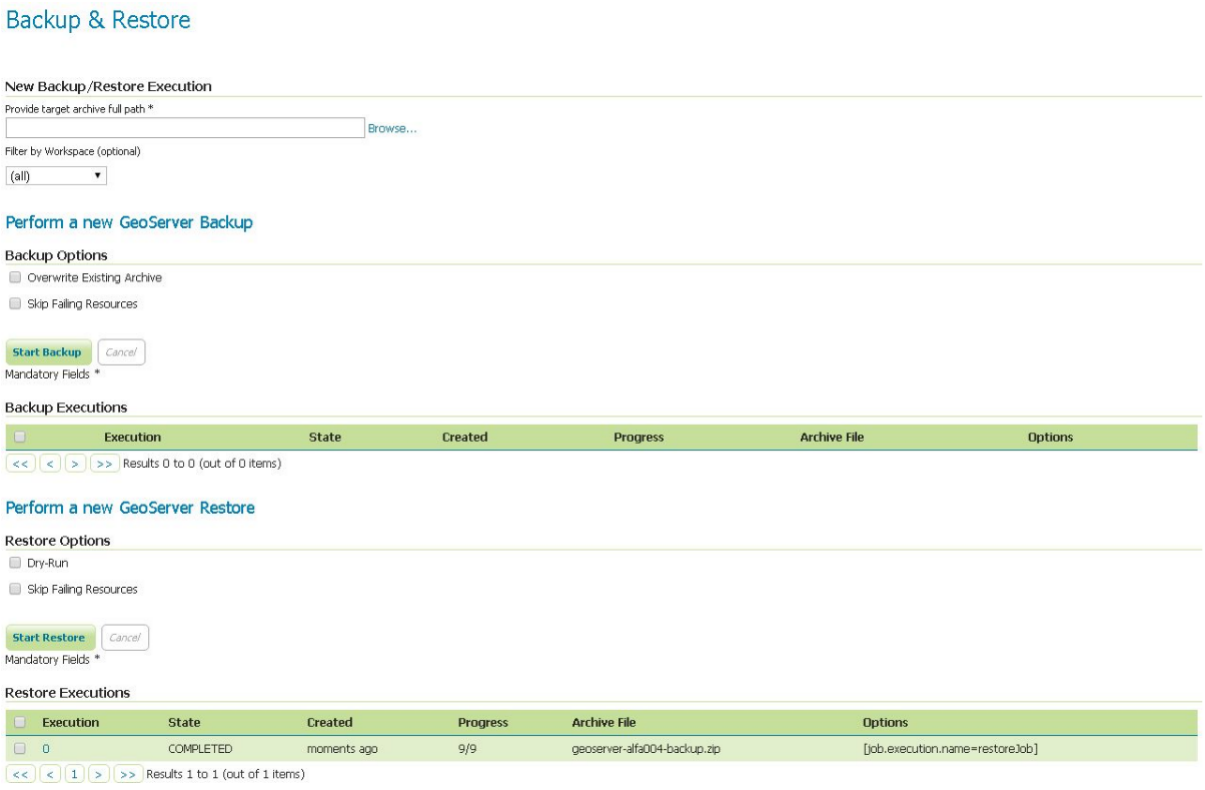
Note: Please notice that while performing a backup or restore task, GeoServer won't allow users to access other sections by locking the catalog and configuration until the process has finished. Although it is always possible to stop or abandon a backup or restore procedure.

At the end of the backup, the user will be redirected to an `Execution Summary` page

The same page can be accessed also later by clicking an execution link from the main page.

Note: Please notice that the list of executions is not persisted and therefore it will be reset after a GeoServer container **restart**.

At the bottom of the `Execution Details` page, it's possible to download the `.zip` archive directly by clicking on the `Download Archive File` link.



In case some running exceptions or warning have been catched by the process, they will be presented on the execution summary. The `Error Detail Level` allows to inspect the causes by exposing the stack trace for each of them.

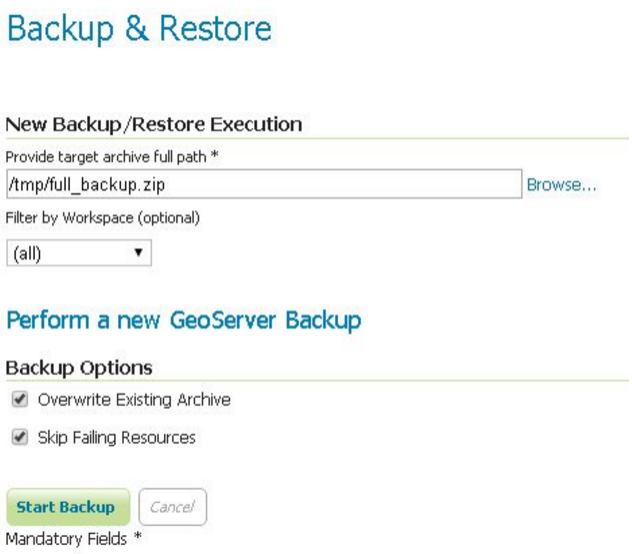
Restoring via UI

The steps are almost the same of the backup. Just select the `.zip` archive full path before launching the restore process.

Warning: Please notice that a **non-dry-run restore** will lose all your current GeoServer configuration by replacing it with the new one, so be careful and be sure to backup everything before starting a restore.

DRY-RUN RESTORE

`Dry Run` option allows a user to **test** a `.zip` archive before actually performing a full restore.



Note: Please notice that the dry run should always being executed when trying to restore a new configuration.

A **failing** restore dry-run will appear like this

If some exception occurs, it will be listed on the execution summary page. The original cause can be inspected by rising up the errors details level and refreshing

Saving/restoring only specific workspaces

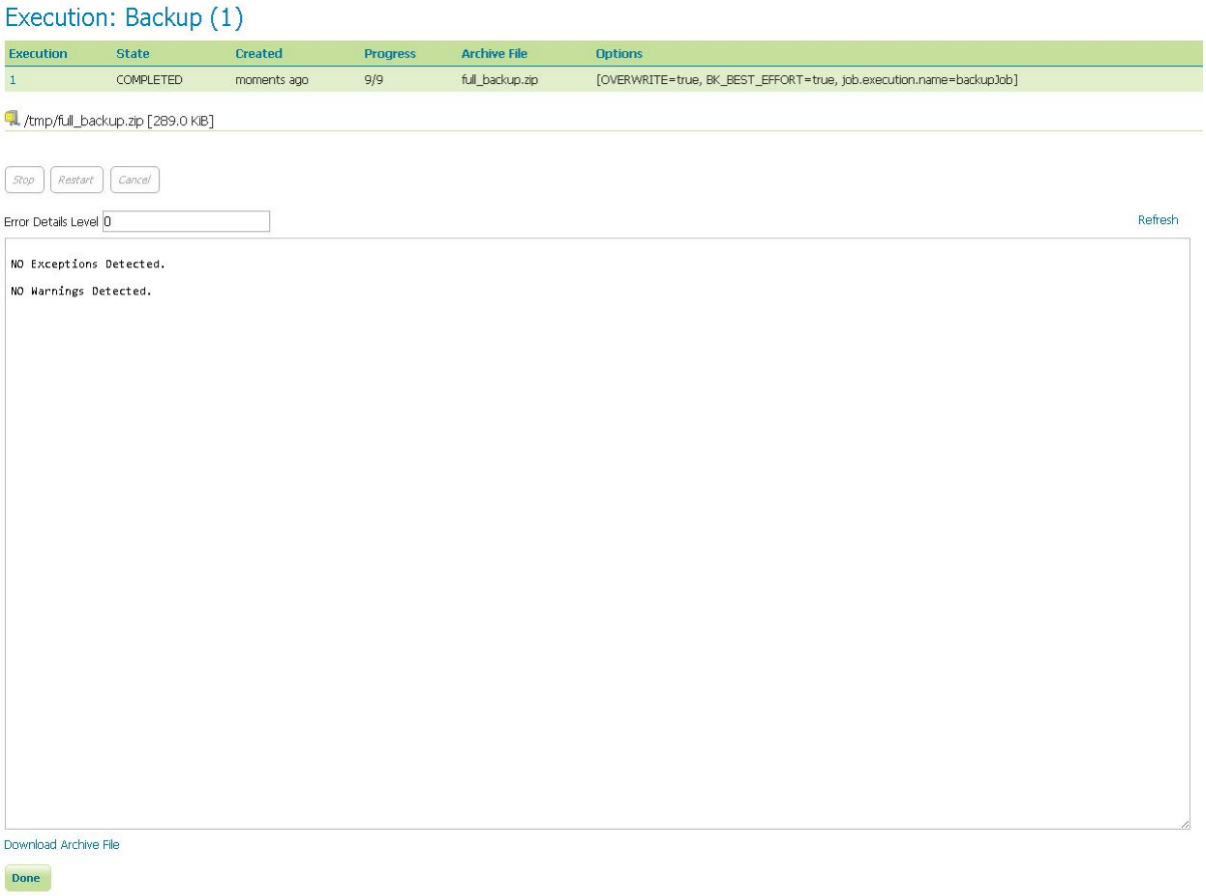
It is possible to backup or restore only a subset of the available workspaces in the catalog. From the WEB interface is currently possible to select all or just one workspace to backup/restore

Through the REST APIs it is possible to filter out also more than one workspaces as explained in the next sections.

Note: Please notice that from a backup archive containing filtered workspaces won't be possible to restore also the missing ones. In order to do that it is advisable to backup the whole catalog and then restore only the workspaces needed.

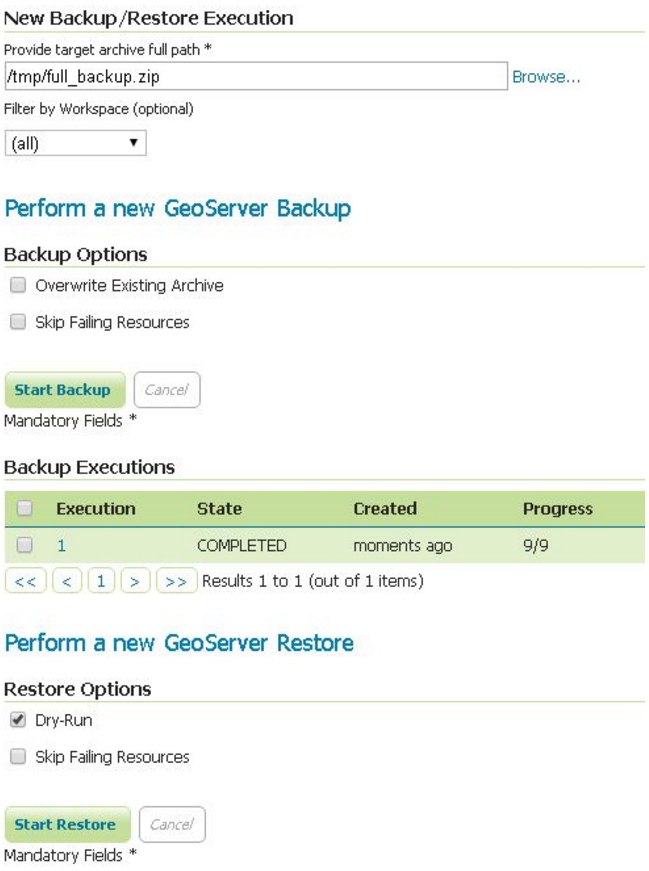
16.24.3 Usage Via GeoServer's REST API

The Backup and Restore REST api consists of a few resources meant to used in an asynchronous fashion:


Execution: Backup (1)
Execution
State
Created
Progress
Archive File
Options
1
COMPLETED
moments ago
9/9
full_backup.zip
[OVERWRITE=true, BK_BEST_EFFORT=true, job.execution.name=backupJob]
/tmp/full_backup.zip [289.0 KiB]
Stop
Restart
Cancel
Error Details Level 0
Refresh
NO Exceptions Detected.
NO Warnings Detected.
Download Archive File
Done


Download Archive File
Done


192.168.1.105:8080/geoserver/web/wicket/bookmarkable/org.geoserver.backuprest
full_backup.zip


New Backup/Restore Execution
Provide target archive full path *
/tmp/full_backup.zip
Browse...
Filter by Workspace (optional)
(all)
Perform a new GeoServer Backup
Backup Options
Overwrite Existing Archive
Skip Failing Resources
Start Backup
Cancel
Mandatory Fields *
Backup Executions
Execution
State
Created
Progress
1
COMPLETED
moments ago
9/9
<< < 1 > >> Results 1 to 1 (out of 1 items)
Perform a new GeoServer Restore
Restore Options
Dry-Run
Skip Failing Resources
Start Restore
Cancel
Mandatory Fields *

Resource	Method	Parameters and Notes
/rest/br/backup/	POST	Post a JSON/XML document with the backup parameters, see below
/rest/br/backup/backupId	GET	Returns a json/xml representation of the backup operation. See below
/rest/br/backup/backupId	DELETE	Cancels the backup operation
/rest/br/restore	POST	Post a JSON/XML document with the restore parameters, see below
/rest/br/restore/restoreId	GET	Returns a json/xml representation of the backup operation, see below
/rest/br/restore/restoreId	DELETE	Cancels the restore operation

Usage Example

We are going to use the command line tool cURL to send HTTP REST requests to GeoServer.

The `/rest/br/backup/` and `/rest/br/restore` endpoints accept an optional format suffix that allows the Backup / Restore archive to be streamed to / from the client instead of being written on / read from the file system.

Initiate a Backup

Prepare a file containing with a JSON object representing the Backup procedure configuration.

`backup_post.json`

```
{
   "backup":{
      "archiveFile":"/home/sg/BackupAndRestore/test_rest_1.zip",
      "overwrite":true,
      "options":{
      }
   }
}
```

In this case we did not specify any options in the backup configuration so default values will be used.

Available options are:

1. `BK_BEST_EFFORT`: Skip any failing resources and proceed with the backup procedure
2. `BK_PARAM_PASSWORDS`: Whether outgoing store passwords should be parameterized in the backup. With this option set all store passwords will be replaced with a token that looks like *${workspaceName:storeName.passwd.encryptedValue}*
3. `BK_SKIP_SECURITY`: _Experimental_. This will exclude security settings from the backup.
4. `BK_SKIP_SETTINGS`: _Experimental_. This will attempt to exclude most global settings from the backup, as well as security settings.

Also an optional `Filter` can be passed to restrict the scope of the restore operation to a list of workspaces.

For example

```
{
   "backup":{
      "archiveFile":"/home/sg/BackupAndRestore/test_rest_1.zip",
"overwrite":true,
      "options":{
```

```
        "option": ["BK_BEST_EFFORT=true"]
      },
"filter": "name IN ('topp','geosolutions-it')"
   }
}
```

Backup procedure will be initiated.

Here is a sample response

```
HTTP/1.1 201 Created
Date: Mon, 01 Aug 2016 14:35:44 GMT
Location: http://mygeoserver/geoserver/rest/br/backup/1
Server: Noelios-Restlet-Engine/1.0..8
Content-Type: application/json
Transfer-Encoding: chunked

{
   "backup":{
      "totalNumberOfSteps":9,
      "execution":{
         "id":1,
         "version":1,
         "stepExecutions":{
            "@class":"java.util.concurrent.CopyOnWriteArraySet"
         },
         "status":[
            "STARTED"
         ],
         "startTime":"2016-08-01 14:35:44.802 UTC",
         "createTime":"2016-08-01 14:35:44.798 UTC",
         "lastUpdated":"2016-08-01 14:35:44.803 UTC",
         "exitStatus":{
            "exitCode":"UNKNOWN",
            "exitDescription":""
         },
         "progress":"1\/9"
      },
      "options":{
         "@class":"synchList",
         "option":[
            "OVERWRITE=true"
         ]
      },
      "warningsList":{
         "@class":"synchList"
      },
      "archiveFile":{
         "@class":"resource",
         "$":"\/home\/sg\/BackupAndRestore\/test_rest_1.zip"
      },
      "overwrite":true
   }
}
```

At the end of the backup procedure you'll be able to download the generated archive to your local file system by making an HTTP GET request to the same endpoint, using the **backup ID** as above and adding the `.zip` at the end

```
curl -u "admin:geoserver" -i -X GET  "http://mygeoserver/geoserver/rest/br/
↪backup/1.zip" -o 1.zip
```

Query status of Backup executions

Status of the operation can be queried making an HTTP GET request to the location listed in the response,

```
http://mygeoserver/geoserver/rest/br/backup/$ID.{json/xml}
```

Replace `$ID` with the **ID** of the backup operation you'd like to inspect.

```
curl -u "admin:geoserver" http://mygeoserver/geoserver/rest/br/backup/1.json
```

or

```
curl -u "admin:geoserver" http://mygeoserver/geoserver/rest/br/backup/1.xml
```

GeoServer will respond with the status of the backup job corresponding to that ID

Here you are able to see the status of all the steps involved in the backup procedure with creation time, start time, end time, exit status etc.

Cancel a Backup

Cancel an in progress Backup by sending an HTTP DELETE request with the ID of the task

```
curl -v -XDELETE -u "admin:geoserver" http://mygeoserver/geoserver/rest/br/
↪backup/$ID
```

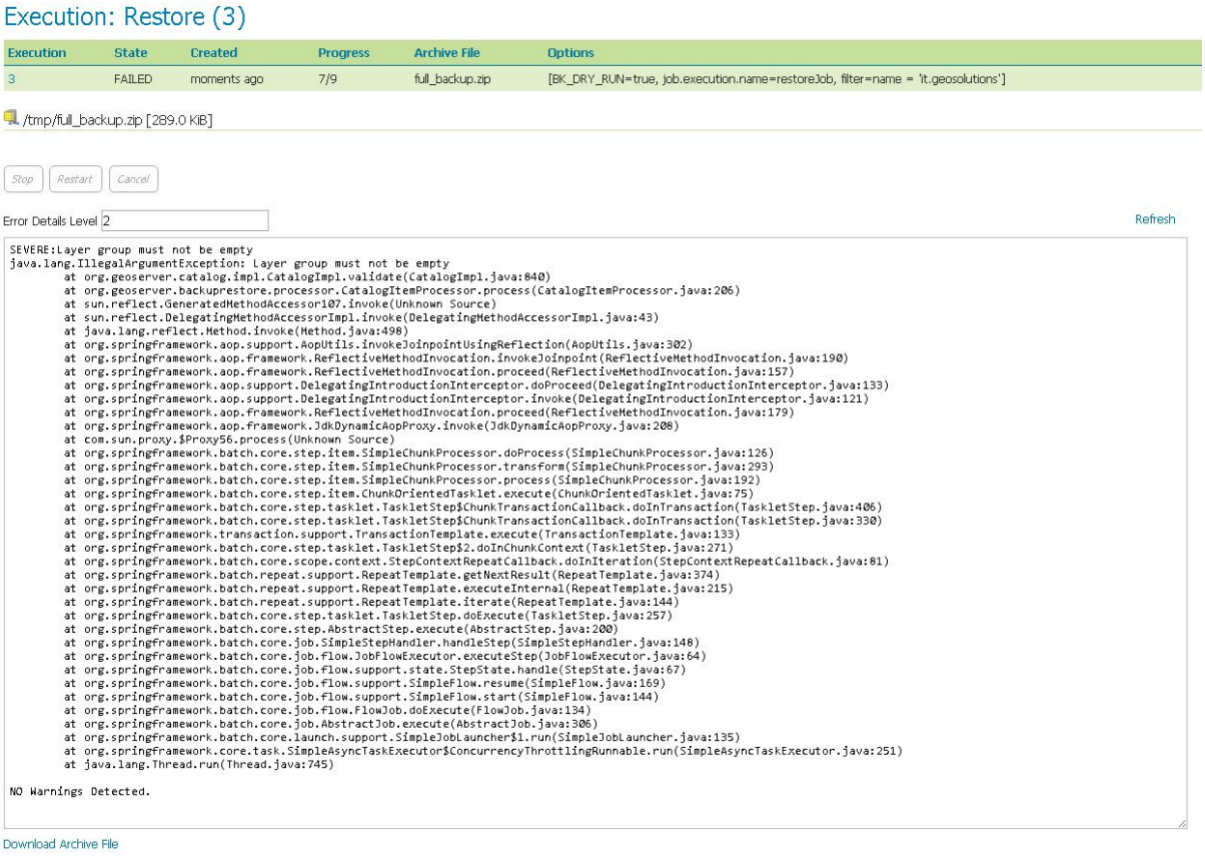
Replace `$ID` with the **ID** of the backup operation you'd like to cancel.

Initiate a Restore

Prepare a file with a JSON object representing the Restore procedure configuration

```
restore_post.json
```

```
{
   "restore":{
      "archiveFile":"/home/sg/BackupAndRestore/test_rest_1.zip",
      "options":{
      }
   }
}
```



In this case we did not specify any options in the restore configuration so default values will be used.

Available Options are:

1. `BK_DRY_RUN`: Only test the archive do not persist the restored configuration
2. `BK_BEST_EFFORT`: Skip any failing resources and proceed with the restore procedure
3. `BK_PASSWORD_TOKENS`: A comma separated list of equal sign separated key/values to be replaced in data store passwords in an incoming backup. For example

```
BK_PASSWORD_TOKENS=${workspace:store1.passwd.encryptedValye}=foo,$
↪{workspace:store2.passwd.encryptedValue}=bar
```

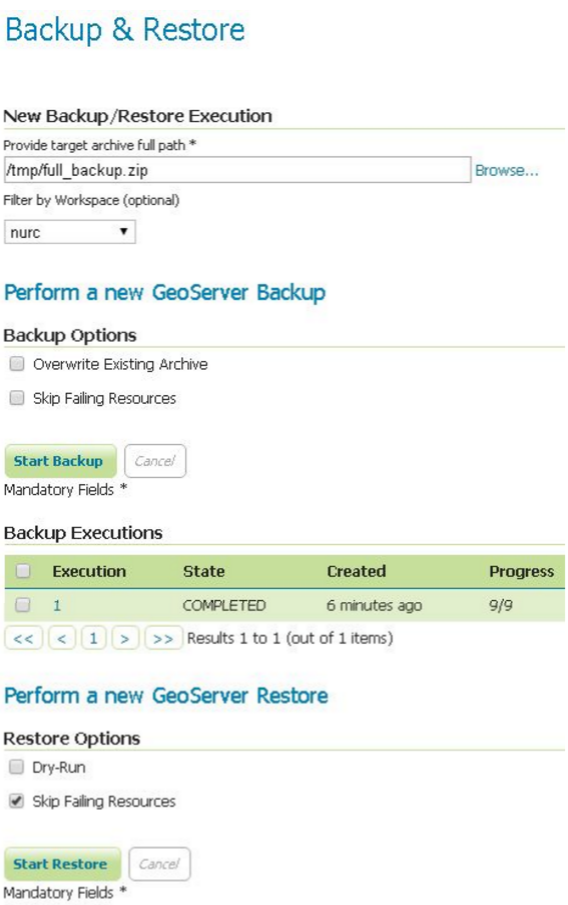
4. `BK_SKIP_SECURITY`: _Experimental_. This will exclude security settings from the restore. Default: *false*.
5. `BK_SKIP_SETTINGS`: _Experimental_. This will attempt to exclude most global settings from the backup, as well as security settings. Default: *false*

#. `BK_PURGE_RESOURCES`: _Experimental_. This will skip deleting incoming resources where possible. In particular, existing workspaces will not be deleted during the restore. Default: *true*

Also an optional `Filter` can be passed to restict the scope of the restore operation to a list of workspaces.

For example

```
{
   "restore":{
      "archiveFile":"/home/sg/BackupAndRestore/test_rest_1.zip",
```


Backup & Restore
New Backup/Restore Execution
Provide target archive full path *
/tmp/full_backup.zip
Browse...
Filter by Workspace (optional)
nurc
Perform a new GeoServer Backup
Backup Options
Overwrite Existing Archive
Skip Failing Resources
Start Backup
Cancel
Mandatory Fields *
Backup Executions
Execution
State
Created
Progress
1
COMPLETED
6 minutes ago
9/9
<< < 1 > >> Results 1 to 1 (out of 1 items)
Perform a new GeoServer Restore
Restore Options
Dry-Run
Skip Failing Resources
Start Restore
Cancel
Mandatory Fields *

```
      "options":{
        "option": ["BK_DRY_RUN=true"]
      },
"filter": "name IN ('topp','geosolutions-it')"
   }
}
```

If `archiveFile` is specified, the archive specified on that path of the remote file system will be used to initiate the restore procedure. Otherwise you're the archive needs to be uploaded from your local system.

Then make a POST HTTP request to GeoServer's REST interface endpoint for the restore procedure

```
curl -u "admin:geoserver" -i -H "Content-Type: application/json" -X POST --
↪data @restore_post.json http://mygeoserver/geoserver/rest/br/restore/
```

Restore procedure will be initiated.

Here is a sample response

```
HTTP/1.1 201 Created
Date: Mon, 01 Aug 2016 15:07:29 GMT
Location: http://mygeoserver/geoserver/rest/br/restore/2
Server: Noelios-Restlet-Engine/1.0..8
Content-Type: application/json
Transfer-Encoding: chunked

{
   "restore":{
      "totalNumberOfSteps":9,
      "execution":{
         "id":2,
         "version":1,
         "stepExecutions":{
            "@class":"java.util.concurrent.CopyOnWriteArraySet"
         },
         "status":[
            "STARTED"
         ],
         "startTime":"2016-08-01 15:07:29.398 UTC",
         "createTime":"2016-08-01 15:07:29.393 UTC",
         "lastUpdated":"2016-08-01 15:07:29.398 UTC",
         "exitStatus":{
            "exitCode":"UNKNOWN",
            "exitDescription":""
         },
         "progress":"0\/9"
      },
      "options":{
         "@class":"synchList"
      },
      "warningsList":{
         "@class":"synchList"
      },
      "archiveFile":{
         "@class":"resource",
         "$":"\/home\/sg\/BackupAndRestore\/test_rest_1.zip"
      }
   }
}
```

To upload the archive from our local system instead, omit the archiveFile parameter in the JSON object and pass the `--upload-file` parameter to cURL:

`restore_post.json`

```
{
   "restore":{
      "options":{
      },
   }
}
```

```
curl -u "admin:geoserver" -i -H "Content-Type: application/json" --upload-
↪file "archive_to_restore.zip" -X POST --data @restore_post.json http://
↪localhost:8081/geoserver/rest/br/restore/
```

Local `archive_to_restore.zip` archive will be uploaded and used by the restore procedure.

Query for status of Restore operations

`http://mygeoserver/geoser/restore/$ID.{json/xml}`

```
{
   "restore":{
      "execution":{
         "hash":2,
         "key":{
            "@class":"long",
            "$":"2"
         },
         "val":{
```


```
            "@class":"restore",
            "totalNumberOfSteps":9,
            "execution":{
               "id":2,
               "version":2,
               "stepExecutions":{
                  "@class":"java.util.concurrent.CopyOnWriteArraySet",
                  "step":[
                     {
                        "name":"restoreNamespaceInfos",
                        "status":"COMPLETED",
                        "exitStatus":{
                           "exitCode":"COMPLETED",
                           "exitDescription":""
                        },
                        "startTime":"8\/1\/16 3:07 PM",
                        "endTime":"8\/1\/16 3:07 PM",
                        "lastUpdated":"8\/1\/16 3:07 PM",
                        "parameters":{
                           "input.file.path":"file:\/\/\/opt\/tomcat-
↪geoserver-2.9.x\/temp\/tmpbbe2388a-f26d-4f26-a20f-88c653d88aec",
                           "time":1470064049392
                        },
                        "readCount":1,
                        "writeCount":1,
                        "failureExceptions":""
                     },
```

```
                    ...
                     {
                        "name":"restoreGeoServerSecurityManager",
                        "status":"COMPLETED",
                        "exitStatus":{
                           "exitCode":"COMPLETED",
                           "exitDescription":""
                        },
                        "startTime":"8\/1\/16 3:07 PM",
                        "endTime":"8\/1\/16 3:07 PM",
                        "lastUpdated":"8\/1\/16 3:07 PM",
                        "parameters":{
                           "input.file.path":"file:\/\/\/opt\/tomcat-
↪geoserver-2.9.x\/temp\/tmpbbe2388a-f26d-4f26-a20f-88c653d88aec",
                           "time":1470064049392
                        },
                        "readCount":0,
                        "writeCount":0,
                        "failureExceptions":""
                     }
                  ]
               },
               "status":"COMPLETED",
               "startTime":"2016-08-01 15:07:29.398 UTC",
               "createTime":"2016-08-01 15:07:29.393 UTC",
               "endTime":"2016-08-01 15:07:30.356 UTC",
               "lastUpdated":"2016-08-01 15:07:30.772 UTC",
               "exitStatus":{
                  "exitCode":"COMPLETED",
                  "exitDescription":""
               },
               "progress":"9\/9"
            },
            "options":{
               "@class":"synchList"
            },
            "warningsList":{
               "@class":"synchList"
            },
            "archiveFile":{
               "@class":"resource",
               "$":"\/home\/sg\/BackupAndRestore\/test_rest_1.zip"
            }
         }
      }
   ...
```


Here you are able to see the status of all the steps involved in the restore procedure with creation time, start time, end time, exit status etc.

Cancel a Restore

Cancel an in progress Restore by sending an HTTP DELETE request:

```
curl -v -XDELETE -u "admin:geoserver" http://mygeoserver/geoserver/rest/br/
↪restore/$ID
```

Replace `$ID` with the **ID** of the restore operation you'd like to cancel.

16.24.4 Backup and Restore Extension for the management of ImageMosaic indexers

Introduction

ImageMosaics CoverageStores make use of several `.properties` files instructing the reader on how to create the mosaic index.

What we want to achieve is to allow the GeoServer Backup & Restore module to *inject* environment properties on indexers allowing the ImageMosaic to be automatically ported among different environments.

Technical Details

The GeoServer Backup & Restore module actually provides an extension point on reading / writing allowing GeoServer to handle additional resources related to a particular `ResourceInfo`.

The interfaces

```
public interface CatalogAdditionalResourcesWriter<T> {

    public boolean canHandle(Object item);

    public void writeAdditionalResources(Backup backupFacade, Resource base,␣
↪T item)
            throws IOException;

}
```

```
public interface CatalogAdditionalResourcesReader<T> {

    public boolean canHandle(Object item);

    public void readAdditionalResources(Backup backupFacade, Resource base,␣
↪T item)
            throws IOException;

}
```

Is invoked by the `CatalogFileWriter` (when doing a Backup) and the `CatalogItemWriter` (when doing a Restore) after a successful write of the resource configuration on the, respectively, target backup folder and in-memory catalog.

The idea is the following one *allowing the CatalogItemWriter to*:

1. Restore the ImageMosaic Indexer Properties injecting environment properties
2. Check if the Mosaic index physically exist and if not create an empty one

In order to do that we envisage the following technical approach

On a **BACKUP** operation

1. The Additional Resource Writer checks if the `ResourceInfo` is an ImageMosaic Coverage Store.
2. The Additional Resource Writer looks for `*.template` files on the ImageMosaic index directory. It must store them into the zip archive by reading the path from the Coverage Store.
3. The Additional Resource Writer stores the `*.template` along with the `*.properties` files on the target backup folder. Same as above.

On a **RESTORE** operation

1. The Additional Resource Reader checks if the `ResourceInfo` is an ImageMosaic Coverage Store.
2. The Additional Resource Reader looks for `*.template` files on the ImageMosaic index directory. It will try to restore them by using the path read from the Coverage Store configuration.
3. The Additional Resource Reader overwrites the `*.properties` files by resolving all the environment properties declared on the templates.
4. The Additional Resource Reader checks if the empty mosaic must be created or not.

16.24.5 Use Cases

Database vs. Shapefile based indexer

When using a DataBase as backend storage for the mosaic index, a `datastore.properties` file is present on the mosaic folder containing the connection parameters.

In case the user wants to parametrize this, he must create a `.template` datastore properties file containing all the properties of the original one but using placemarks as parametric values.

As an instance:

```
host=${mosaic1.jdbc.host}
port=${mosaic1.jdbc.port}
...
```

The backup and restore extension will save on the archive both the original `.properties` and the `.template`

When restoring, the extension will overwrite the `.properties` by using the `.template` and substituting the placemarks with the correct environment property values.

When using a shapefile as backend for the index the shapefile itself will be created once again by the mosaic when performing the first harvest operation.

Database Connection Parameters vs. JNDI

This use case is similar to the previous one, except for the fact that instead of parameters like host and port we will have a parametric JNDI name.

Indexer files and regex

The approach will be exactly the same of the `datastore.properties`.

Is is worth notice that the backup extension will overwrite only the files having a corresponding `.template` prototype.

Granules stored on the same mosaic folder vs. absolute path

This won't impact the backup and restore at all, since it will never dumps data into the final archive.

It is important, however, that the absolute paths are parametric similar to the connection parameters explained above.

Dealing with non-existing indexes on the target restored environment

It is possible that when restoring the ImageMosaic the index does not exist on the target environment.

The backup and restore extension should perform a double check once restored the `datastore.properties` file trying to access the index store.

1. In case of failure, i.e. the extension cannot connect to the datastore, the resource will fail.
2. In case the datastore is accessible but the index does not exist, the plugin will create an empty mosaic on the catalog instead of failing.

16.25 OneLogin Authentication Filter

This plugin adds to GeoServer the support for SAML based Single Sign On (SSO), a process that allows users to authenticate themselves against an external Identity Provider (such as OneLogin) rather than obtaining and using a separate username and password handled by GeoServer.

What you will need is an account in OneLogin: https://www.onelogin.com/ that will handle the sign-in process and will eventually provide the authentication credentials of your users to GeoServer.

GeoServer users authenticated through OneLogin are handled from OneLogin and any change performed on the account is used by GeoServer. The only user data that is necessary for GeoServer is a unique identifier for each user.

User's email is used by default as a unique identifier for each user. GeoServer does not store passwords.

16.25.1 OneLogin Configuration

Actually GeoServer is not present within the OneLogin application catalog so we can use the OneLogin SAML test connector. For more details about configuring the SAML Test Connector follow the guide at:

https://support.onelogin.com/hc/en-us/articles/202673944-How-to-Use-the-OneLogin-SAML-Test-Connector

In the example we assume that GeoServer URL is http://localhost:8080/geoserver, if you have a specific domain for geoserver use it instead.

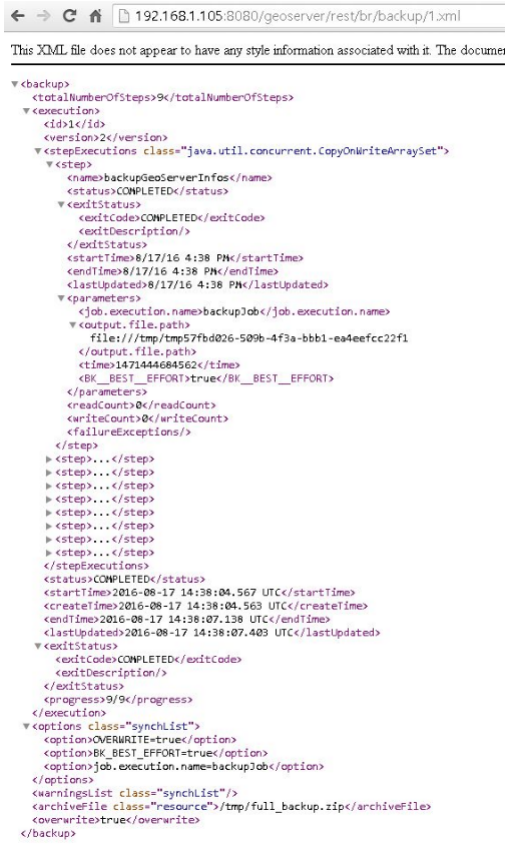
On the SAML Test Connector (IdP) configuration page, use the following values as parameters:

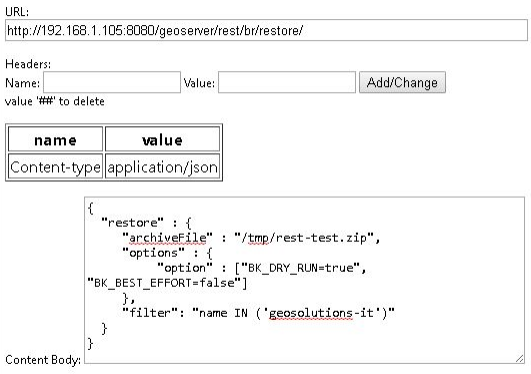
Parameter	Value
RelayState	empty
Audience	GeoServer
Recipient	http://localhost:8080/geoserver/saml/SSO
ACS URL Validator	^http://localhost:8080/geoserver/saml/SSO$
ACS URL	http://localhost:8080/geoserver/saml/SSO
Single Logout URL	http://localhost:8080/geoserver/saml/SingleLogout

Then, write down the Metadata URL you found in the `Issuer URL` field of the SSO Configuration page:

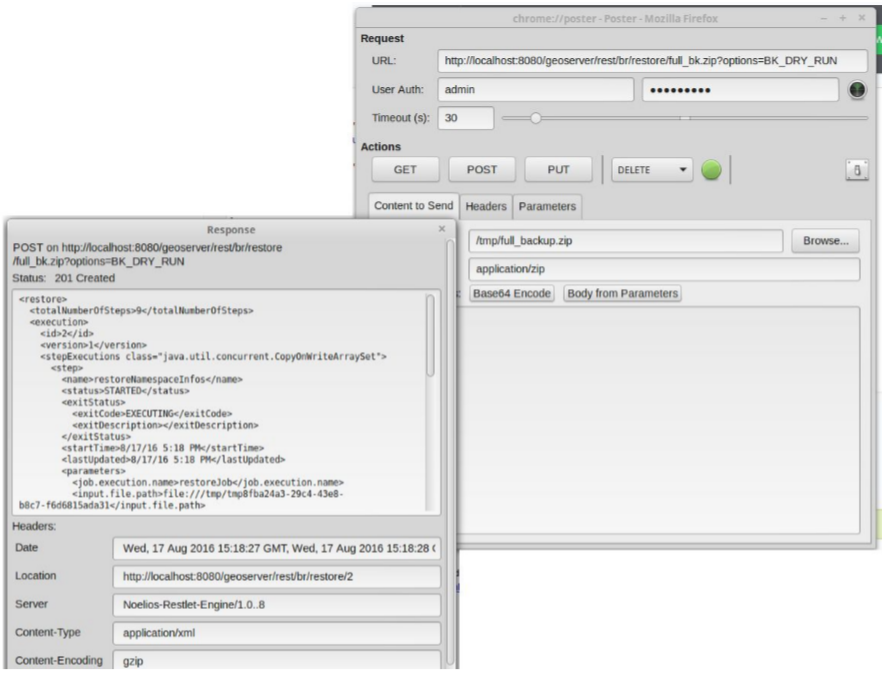
16.25.2 Configuring the OneLogin Authentication Filter

1. Start GeoServer and login to the web admin UI as the admin user.
2. Click the Authentication link located under the Security section of the navigation sidebar.


192.168.1.105:8080/geoserver/rest/br/backup/1.xml
This XML file does not appear to have any style information associated with it. The docume
<backup>
<totalNumberOfSteps>9</totalNumberOfSteps>
<execution>
<id>1</id>
<version>2</version>
<stepExecutions class="java.util.concurrent.CopyOnWriteArraySet">
<step>
<name>backupGeoServerInfos</name>
<status>COMPLETED</status>
<exitStatus>
<exitCode>COMPLETED</exitCode>
<exitDescription/>
</exitStatus>
<startTime>8/17/16 4:38 PM</startTime>
<endTime>8/17/16 4:38 PM</endTime>
<lastUpdated>8/17/16 4:38 PM</lastUpdated>
<parameters>
<job.execution.name>backupJob</job.execution.name>
<output.file.path>
file:///tmp/tmpS7fbd026-509b-4f3a-bbb1-ea4eefcc22f1
</output.file.path>
<time>1471444684562</time>
<BK__BEST__EFFORT>true</BK__BEST__EFFORT>
</parameters>
<readCount>0</readCount>
<writeCount>0</writeCount>
<failureExceptions/>
</step>
<step>...</step>
<step>...</step>
<step>...</step>
<step>...</step>
<step>...</step>
<step>...</step>
<step>...</step>
<step>...</step>
</stepExecutions>
<status>COMPLETED</status>
<startTime>2016-08-17 14:38:04.567 UTC</startTime>
<createTime>2016-08-17 14:38:04.563 UTC</createTime>
<endTime>2016-08-17 14:38:07.138 UTC</endTime>
<lastUpdated>2016-08-17 14:38:07.403 UTC</lastUpdated>
<exitStatus>
<exitCode>COMPLETED</exitCode>
<exitDescription/>
</exitStatus>
<progress>9/9</progress>
</execution>
<options class="synchList">
<option>OVERWRITE=true</option>
<option>BK_BEST_EFFORT=true</option>
<option>job.execution.name=backupJob</option>
</options>
<warningsList class="synchList"/>
<archiveFile class="resource">/tmp/full_backup.zip</archiveFile>
<overwrite>true</overwrite>
</backup>


URL:
http://192.168.1.105:8080/geoserver/rest/br/restore/
Headers:
Name:
Value:
Add/Change
value '##' to delete
name
value
Content-type
application/json
{
"restore" : {
"archiveFile" : "/tmp/rest-test.zip",
"options" : {
"option" : ["BK_DRY_RUN=true",
"BK_BEST_EFFORT=false"]
},
"filter": "name IN ('geosolutions-it')"
}
}
Content Body:

3. Scroll down to the Authentication Filters panel and click the Add new link.



4. Click the OneLogin link.

5. Fill in the fields of the settings form as follows; you can use GeoServer role sources to assign a specific group or role to OneLogin user. OneLogin user email must be mapped to GeoServer user name.

Parameter	Value
Name	OneLogin (or any other unique name)
Identity provider identifier	Same value of Audience field on OneLogin configuration panel
SAML metadata URL	Put here the Metadata URL copied from the OneLogin configuration panel

6. Update the filter chains by adding the new OneLogin filter.

7. Select the OneLogin Filter for each filter chain you want to protect with OneLogin (for example web)

16.25.3 Testing with OneLogin

1. Navigate to the GeoServer home page and log out of the admin account.
2. Try to login again, you should be able now to see the external OneLogin form.
3. You can manage users using OneLogin panel located at https://admin.us.onelogin.com/users

16.26 WMTS Multidimensional

This module implements the WMTS multidimensional domain discovery extensions as proposed in this document.

This documentation will be very practical, is highly recommended to read the document linked above for a better understanding of the implemented multidimensional domain discovery extensions.

← SAML Test Connector (IdP)

MORE ACTIONS SAVE

Info Configuration Parameters Rules SSO Access Users

Enable SAML2.0

Sign on method

SAML2.0

X.509 Certificate

Standard Strength Certificate (2048-bit)

Change | View Details

SAML Signature Algorithm

SHA-1

Issuer URL

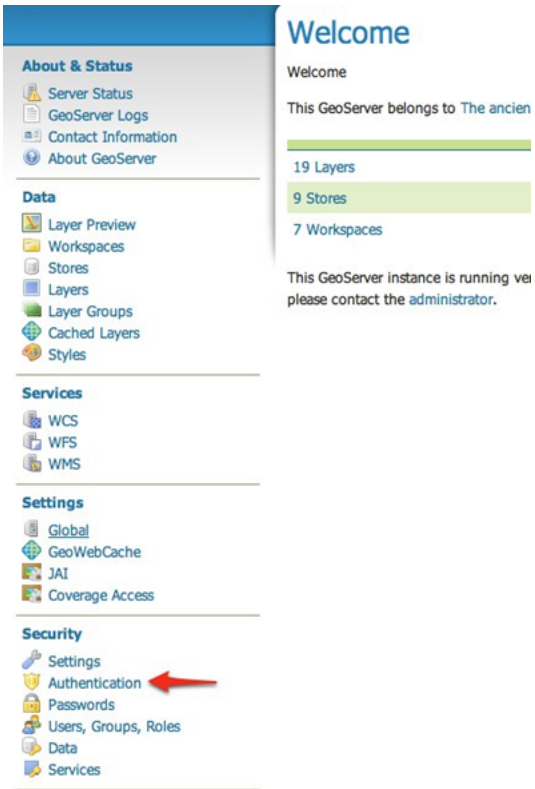
SAML 2.0 Endpoint (HTTP)

SLO Endpoint (HTTP)

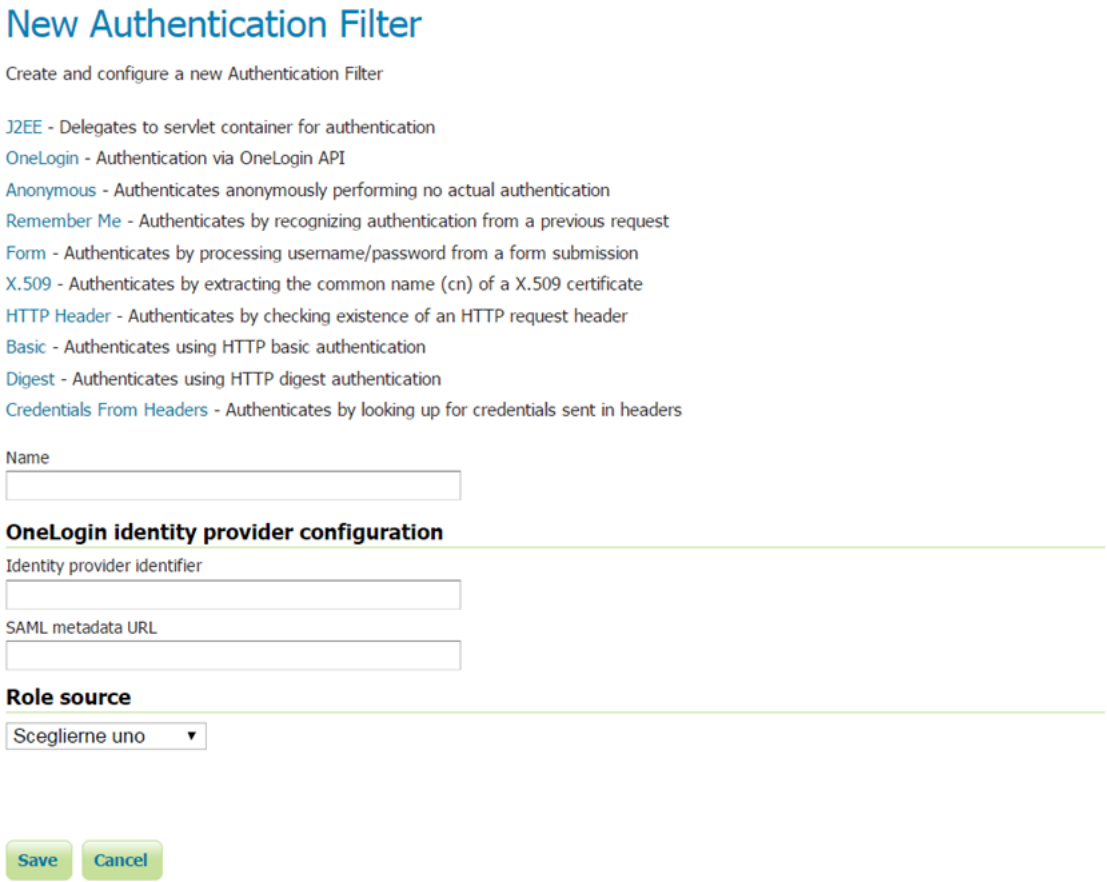
Assumed Sign-In

Allow assumed users to sign into this app

When enabled, admins who assume users can sign into this app with their identity. This setting can only be changed by the account owner. Note that the account owner can also completely disable the assume feature under Account -> Settings.




Authentication
Authentication providers and settings
Logout settings
Redirect URL after logout (empty,absolute or relative to context root)
/web/
SSL settings
SSL Port (default is 443)
443
Authentication Filters
Add new
Remove selected


New Authentication Filter
Create and configure a new Authentication Filter
J2EE - Delegates to servlet container for authentication
OneLogin - Authentication via OneLogin API
Anonymous - Authenticates anonymously performing no actual authentication
Remember Me - Authenticates by recognizing authentication from a previous request
Form - Authenticates by processing username/password from a form submission
X.509 - Authenticates by extracting the common name (cn) of a X.509 certificate
HTTP Header - Authenticates by checking existence of an HTTP request header
Basic - Authenticates using HTTP basic authentication
Digest - Authenticates using HTTP digest authentication
Credentials From Headers - Authenticates by looking up for credentials sent in headers
Name
OneLogin identity provider configuration
Identity provider identifier
SAML metadata URL
Role source
Sceglierne uno
Save
Cancel

All described operations including is optional parameters and other extensions were implemented, only the the REST interfaces for the domain discovery operations were not implemented.

The `GetFeature` operation only supports the profile GML 3.1 as feature info format ("application/gml+xml; version=3.1") and the `GetHistogram` operation only supports `text/xml` as output format.

This module support well defined dimensions like elevation and time and also custom dimensions.

16.26.1 Installing

This is a community module, which means that it will not be available in the GeoServer official releases and needs to be installed manually.

This module can be installed following these steps:

1. Download this module package from the nightly builds, the module version should match the desired GeoServer version.
2. Extract the contents of the package into the `WEB-INF/lib` directory of the GeoServer installation.

Note: The profile `wmts-multidimensional` can be used to build GeoServer with this module activated, e.g. `mvn clean install -Pwmts-multidimensional -T4 -DskipTests`.

A simple `DescribeDomains` request can be used to test if the module was correctly installed, the request can be made against any layer known by the WMTS service. For example, using the demo layer `tiger:poly_landmarks` shipped with GeoServer:

```
http://localhost:8080/geoserver/gwc/service/wmts?REQUEST=DescribeDomains&Version=1.0.
↪0&Layer=tiger:poly_landmarks&TileMatrixSet=EPSG:4326
```

The result should be similar to the following, this layer doesn't have any domains:

```
<?xml version="1.0" encoding="UTF-8"?><Domains xmlns="http://demo.geo-solutions.it/
↪share/wmts-multidim/wmts_multi_dimensional.xsd" xmlns:ows="http://www.opengis.net/
↪ows/1.1">
  <SpaceDomain>
    <BoundingBox CRS="EPSG:4326" minx="0.0" miny="0.0" maxx="-1.0" maxy="-1.0"/>
  </SpaceDomain>
</Domains>
```

If the module is not correctly installed the result will be an exception saying that this operation is not available:

```
<ExceptionReport version="1.1.0" xmlns="http://www.opengis.net/ows/1.1"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.opengis.net/ows/1.1 http://geowebcache.org/schema/
↪ows/1.1.0/owsExceptionReport.xsd">
  <Exception exceptionCode="OperationNotSupported" locator="request">
    <ExceptionText>describedomains is not implemented</ExceptionText>
  </Exception>
</ExceptionReport>
```

16.26.2 GetCapabilities

The default behavior of WMTS is to list in the capabilities document all the values available in a certain dimension, something like this:

```
<Dimension>
  <ows:Identifier>elevation</ows:Identifier>
  <Default>0.0</Default>
  <Value>0.0</Value>
  <Value>200.0</Value>
  <Value>400.0</Value>
  <Value>600.0</Value>
  <Value>800.0</Value>
  <Value>1000.0</Value>
  <Value>1200.0</Value>
  <Value>1400.0</Value>
  <Value>1600.0</Value>
  <Value>1800.0</Value>
  <Value>2000.0</Value>
  <Value>3000.0</Value>
  <Value>4000.0</Value>
  <Value>5000.0</Value>
  <Value>6000.0</Value>
  <Value>7000.0</Value>
  <Value>8000.0</Value>
  <Value>9000.0</Value>
</Dimension>
```

This module will instead take into account the presentation mode selected by the user:

With the presentation mode select to `Continuous interval` or `Resolution and interval` we will instead see something like this:

```
<Dimension>
  <ows:Identifier>elevation</ows:Identifier>
  <Default>0.0</Default>
  <Value>0.0--9000.0</Value>
</Dimension>
```

Descriptions for the new introduced operations and associated formats will also be added to the capabilities document.

16.26.3 Operations

This module adds three new operations to the WMTS service that are described in detail in this document:

Operation	Description
DescribeDomains	Describes all the dimension domains in a compact document, along with the restricted bounding box of the two dimensional space intercepted by the request.
GetDomainValues	Allows to page through domain values (supplements DescribeDomains in case the domain has too many values, and the client still wants to get all of them, one page at a time)
GetHistogram	Given a scattered domain description and an interval, this operation divides the interval in regular buckets, and provides an item count for each bucket.
GetFeature	Enumerate the actual dimension possible values combinations, returns a list of features along with dimension values using the same formats as the feature info operation ("application/gml+xml; version=3.1").

Note that currently there is no restful implementations of this operations.

DescribeDomains

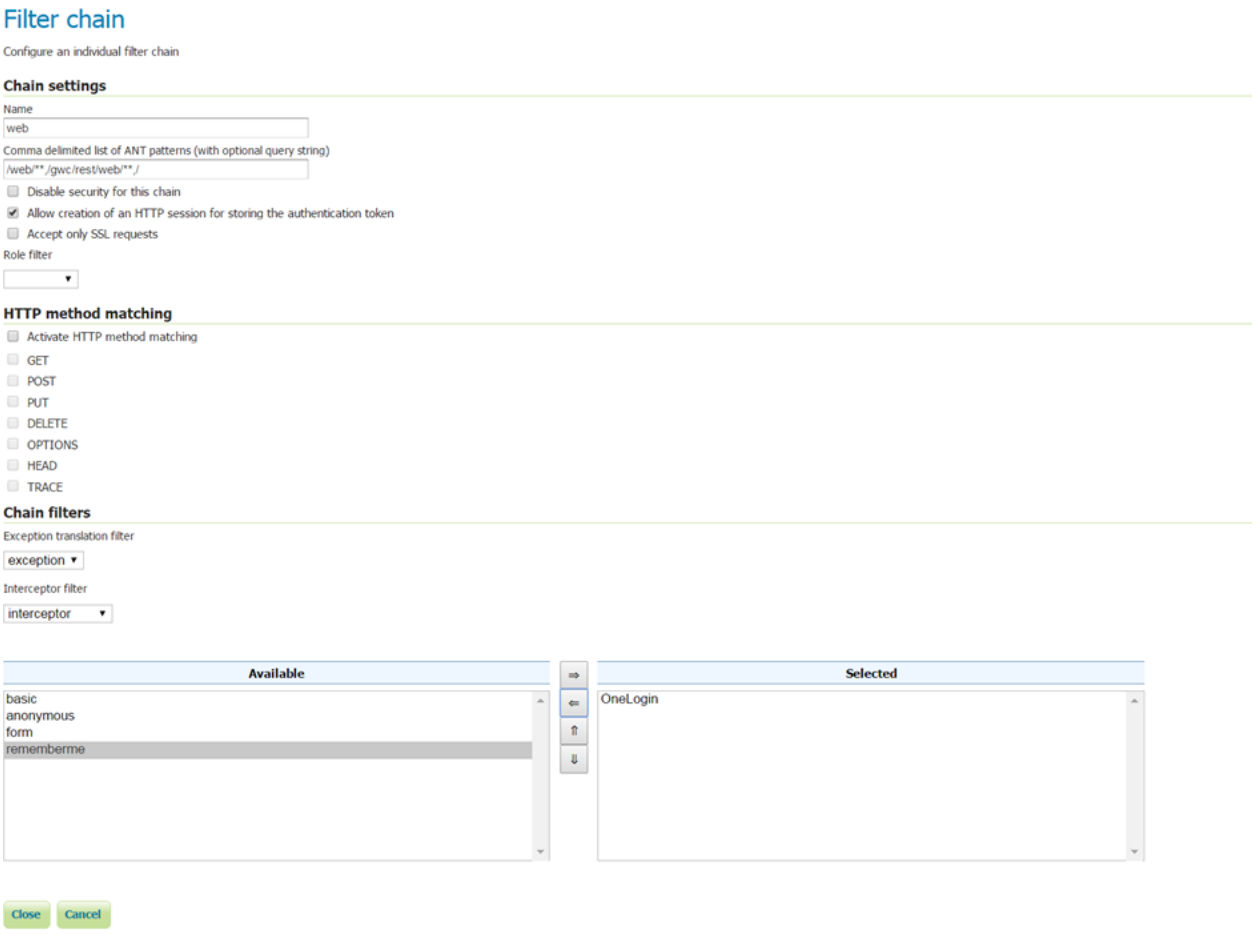
This operation is useful to understand which domains are available in our layer dimensions and how they relate to each other. The parameters available for this operation are:

Name	Mandatory	Description
Service=WMTS	Yes	Service type identifier
Request=DescribeDomains	Yes	Operation name
Version=1.0.0	Yes	Standard and schema version for this operation
Layer	Yes	Layer identifier
TileMatrixSet	Yes	Tile matrix set identifier
bbox=minx,miny,maxx,maxy	No	Bounding box corners (lower left, upper right) in CRS units
DimensionIdentifier	No	At most one per dimension, a range described as min/max, restricting the domain of this dimension
Domains	No	A comma separated list of domain names to be returned, in case only a subset is required. The space domain is identified by "bbox".
ExpandLimit	No	A numerical value, greater or equal to zero. If the number of unique domain values is below `ExpandLimit` then the domain with be represented in full, as a comma separated list of values, otherwise in compact form, as `start--end`. The server assumes a built-in limit of 200 in case not specified, and allows client to specify a value up to 10000, values can be tuned via the user interface, in the WMTS panel (server defaults) and on a layer by layer basis.

The `bbox` parameter allows the client to restrict the `DescribeDomains` operation to a certain spatial area, by default the layer extent will be used.

The `DimensionIdentifier` parameter can be used to restrict the domain values of a certain dimension, this is useful to answer questions like which elevations values are available in a specific day.

A simple `DescribeDomains` request will look like this:


Filter chain
Configure an individual filter chain
Chain settings
Name
web
Comma delimited list of ANT patterns (with optional query string)
/web/**,/gwc/rest/web/**,/
Disable security for this chain
Allow creation of an HTTP session for storing the authentication token
Accept only SSL requests
Role filter
HTTP method matching
Activate HTTP method matching
GET
POST
PUT
DELETE
OPTIONS
HEAD
TRACE
Chain filters
Exception translation filter
exception
Interceptor filter
interceptor
Available
basic
anonymous
form
rememberme
Selected
OneLogin
Close
Cancel

```
http://localhost:8080/geoserver/gwc/service/wmts?REQUEST=DescribeDomains&Version=1.0.
↪0&Layer=some_layer&TileMatrixSet=EPSG:4326
```

and the result will be similar to this:

```
<Domains xmlns="http://demo.geo-solutions.it/share/wmts-multidim/wmts_multi_
↪dimensional.xsd" xmlns:ows="http://www.opengis.net/ows/1.1">
  <SpaceDomain>
    <BoundingBox CRS="EPSG:4326"
     maxx="179.875" maxy="89.9375" minx="-180.125" miny="-90.125"/>
  </SpaceDomain>
  <DimensionDomain>
    <ows:Identifier>elevation</ows:Identifier>
    <Domain>0.0,200.0,400.0,600.0,800.0,1000.0</Domain>
    <Size>6</Size>
  </DimensionDomain>
  <DimensionDomain>
    <ows:Identifier>REFERENCE_TIME</ows:Identifier>
    <Domain>2016-02-23T00:00:00.000Z,2016-02-24T00:00:00.000Z</Domain>
    <Size>2</Size>
  </DimensionDomain>
  <DimensionDomain>
    <ows:Identifier>time</ows:Identifier>
    <Domain>2016-02-23T03:00:00.000Z,2016-02-23T06:00:00.000Z</Domain>
    <Size>2</Size>
  </DimensionDomain>
</Domains>
```

From the information above we can see that we have three dimensions `time`, `elevation` and `REFERENCE_TIME` and the respective domains values.

Now let's see how elevations relate to time dimension by asking which elevations under 500.0 meters are available at time 2016-02-23T03:00:00.000Z:

```
http://localhost:8080/geoserver/gwc/service/wmts?REQUEST=DescribeDomains&Version=1.0.
↪0&Layer=some_layer&TileMatrixSet=EPSG:4326&elevation=0/500&time=2016-02-23T03:00:00.
↪000Z
```

the result will be similar to this:

```
<Domains xmlns="http://demo.geo-solutions.it/share/wmts-multidim/wmts_multi_
↪dimensional.xsd" xmlns:ows="http://www.opengis.net/ows/1.1">
  <SpaceDomain>
    <BoundingBox CRS="EPSG:4326"
     maxx="179.875" maxy="89.9375" minx="-180.125" miny="-90.125"/>
  </SpaceDomain>
  <DimensionDomain>
    <ows:Identifier>elevation</ows:Identifier>
    <Domain>200.0</Domain>
    <Size>1</Size>
  </DimensionDomain>
  <DimensionDomain>
    <ows:Identifier>REFERENCE_TIME</ows:Identifier>
    <Domain>2016-02-23T00:00:00.000Z</Domain>
    <Size>1</Size>
  </DimensionDomain>
  <DimensionDomain>
    <ows:Identifier>time</ows:Identifier>
```

```
    <Domain>2016-02-23T03:00:00.000Z</Domain>
    <Size>1</Size>
  </DimensionDomain>
</Domains>
```

So for time 2016-02-23T03:00:00.000Z there is only values measured at 200.0 meters.

In case only the space domain is of interest, the following request will do:

```
http://localhost:8080/geoserver/gwc/service/wmts?REQUEST=DescribeDomains&Version=1.0.
↪0&Layer=some_layer&TileMatrixSet=EPSG:4326&elevation=0/500&time=2016-02-23T03:00:00.
↪000Z&domains=bbox
```

and the result will be similar to this:

```
<Domains xmlns="http://demo.geo-solutions.it/share/wmts-multidim/wmts_multi_
↪dimensional.xsd" xmlns:ows="http://www.opengis.net/ows/1.1">
  <SpaceDomain>
    <BoundingBox CRS="EPSG:4326"
     maxx="179.875" maxy="89.9375" minx="-180.125" miny="-90.125"/>
  </SpaceDomain>
</Domains>
```

GetDomainValues

This operation is useful to page through the values of a given domain, in case the "multidimensional" area of interest is too large for DescribeDomain to return them in a single shot.

Name	Mandatory	Description
Service=WMTS	Yes	Service type identifier
Request=GetDomainValues	Yes	Operation name
Version=1.0.0	Yes	Standard and schema version for this operation
Layer	Yes	Layer identifier
bbox=minx,miny,maxx,maxy	No	Bounding box corners (lower left, upper right) in CRS units
DimensionIdentifier	No	At most one per dimension, a range described as min/max, restricting the domain of this dimension
Domain	Yes	Name of the domain whose values will be returned (one cannot use "bbox", only single value dimensions can be enumerated by GetDomainValues, e.g., time, elevation).
FromValue	No	Sets the beginning of domain enumeration, for paging purposes. It's not included in the result
Sort	No	Can be "asc" or "desc", determines if the enumeration is from low to high, or from high to low
Limit	No	Maximum number of values returned by this call. The server assumes a built-in limit of 1000 in case not specified, and allows client to specify a value up to 10000.

For example, let's say a "elevation" domain has values 1,2,3 and 5, and that we are paging through it by pages of 2 elements. The client will start without providing a "fromValue", and will then continue using the last value of the previous page as a reference:

```
http://localhost:8080/geoserver/gwc/service/wmts?request=GetDomainValues&Version=1.0.
↪0&Layer=sampleLayer&domain=elevation&limit=2
```

```
<DomainValues xmlns="http://demo.geo-solutions.it/share/wmts-multidim/wmts_multi_
↪dimensional.xsd" xmlns:ows="http://www.opengis.net/ows/1.1">
  <ows:Identifier>elevation</ows:Identifier>
  <Limit>2</Limit>
  <Sort>asc</Sort>
  <Domain>1.0,2.0</Domain>
  <Size>2</Size>
</DomainValues>
```

```
http://localhost:8080/geoserver/gwc/service/wmts?request=GetDomainValues&Version=1.0.
↪0&Layer=sampleLayer&domain=elevation&limit=2&fromValue=2
```

```
<DomainValues xmlns="http://demo.geo-solutions.it/share/wmts-multidim/wmts_multi_
↪dimensional.xsd" xmlns:ows="http://www.opengis.net/ows/1.1">
  <ows:Identifier>elevation</ows:Identifier>
  <Limit>2</Limit>
  <Sort>asc</Sort>
  <FromValue>2.0</FromValue>
  <Domain>3.0,5.0</Domain>
  <Size>2</Size>
</DomainValues>
```

```
http://localhost:8080/geoserver/gwc/service/wmts?request=GetDomainValues&Version=1.0.
↪0&Layer=sampleLayer&domain=elevation&limit=2&fromValue=5
```

```
<DomainValues xmlns="http://demo.geo-solutions.it/share/wmts-multidim/wmts_multi_
↪dimensional.xsd" xmlns:ows="http://www.opengis.net/ows/1.1">
  <ows:Identifier>elevation</ows:Identifier>
  <Limit>2</Limit>
  <Sort>asc</Sort>
  <FromValue>5.0</FromValue>
  <Domain></Domain>
  <Size>0</Size>
</DomainValues>
```

For elevations it might not be uncommon to iterate backwards, from the top-most elevation down to the lowest value. The interaction between client and server migth then look as follows:

```
http://localhost:8080/geoserver/gwc/service/wmts?request=GetDomainValues&Version=1.0.
↪0&Layer=sampleLayer&domain=elevation&limit=2&sort=desc
```

```
<DomainValues xmlns="http://demo.geo-solutions.it/share/wmts-multidim/wmts_multi_
↪dimensional.xsd" xmlns:ows="http://www.opengis.net/ows/1.1">
  <ows:Identifier>elevation</ows:Identifier>
  <Limit>2</Limit>
  <Sort>asc</Sort>
  <Domain>5.0,3.0</Domain>
  <Size>2</Size>
</DomainValues>
```

```
http://localhost:8080/geoserver/gwc/service/wmts?request=GetDomainValues&Version=1.0.
↪0&Layer=sampleLayer&domain=elevation&limit=2&fromValue=3&sort=desc
```

```
<DomainValues xmlns="http://demo.geo-solutions.it/share/wmts-multidim/wmts_multi_
↪dimensional.xsd" xmlns:ows="http://www.opengis.net/ows/1.1">
```

```
  <ows:Identifier>elevation</ows:Identifier>
  <Limit>2</Limit>
  <Sort>asc</Sort>
  <FromValue>3.0</FromValue>
  <Domain>2.0,1.0</Domain>
  <Size>2</Size>
</DomainValues>
```

```
http://localhost:8080/geoserver/gwc/service/wmts?request=GetDomainValues&Version=1.0.
↪0&Layer=sampleLayer&domain=elevation&limit=2&fromValue=1&sort=desc
```

```
<DomainValues xmlns="http://demo.geo-solutions.it/share/wmts-multidim/wmts_multi_
↪dimensional.xsd" xmlns:ows="http://www.opengis.net/ows/1.1">
  <ows:Identifier>elevation</ows:Identifier>
  <Limit>2</Limit>
  <Sort>asc</Sort>
  <FromValue>1.0</FromValue>
  <Domain></Domain>
  <Size>0</Size>
</DomainValues>
```

The paging approach might seem odd for those used to using "limit" and "offset". The main reason it's done this way it's performance, paging through unique values via limit and offset means that the data source has to compute and collect the unique values that are not needed (the ones in previous pages) in order to find the ones in the current page. With large domains (typical of time series) this quickly becomes too slow for interactive usage, as one moves forward in the domain.

By giving a starting point, the unneeded data points can be skipped via index and the distinct value computation can be performed only on the current page data, stopping it as soon as the desired number of results has been computed. With an index on the dimension being queries, this results in nearly constant response times, regardless of the page being requested.

GetHistogram

This operation can be used to provide information about the data distribution between the minimum and maximum values of a certain dimension.

The parameters available for this operation are:

Name	Mandatory	Description
Service=WMTS	Yes	Service type identifier
Request=GetHistogram	Yes	Operation name
Version=1.0.0	Yes	Standard and schema version for this operation
Layer	Yes	Layer identifier
TileMatrixSet	Yes	Tile matrix set identifier
BBOX=minx,miny,maxx,maxy	No	Bounding box corners (lower left, upper right) in CRS units
DimensionIdentifier	No	At most one per dimension, a range described as min/max, restricting the domain of this dimension
Histogram	Yes	Name of the dimension for which the histogram will be computed
Resolution	No	Suggested size of the histogram bucket. Cannot be provided for enumerated dimensions, will use the period syntax for time (e.g. PT1H), a number for numeric dimensions, or auto to leave the decision to the server
Format	No	The desired output format, default is text/html.

The parameters common to the `DescribeDomains` operation work as already described above. Currently only the `text/xml` output format is supported.

The following example request the histogram for time dimension with a resolution of 8 hours restricting elevations between 500.0 and 1000.0 meters:

```
http://localhost:8080/geoserver/gwc/service/wmts?REQUEST=GetHistogram&Version=1.0.0&
↪Layer=some_layer&TileMatrixSet=EPSG:4326&histogram=time&resolution=PT8H&
↪elevation=500.0/1000.0
```

and the result will be similar to this:

```
<Histogram xmlns="http://demo.geo-solutions.it/share/wmts-multidim/wmts_multi_
↪dimensional.xsd" xmlns:ows="http://www.opengis.net/ows/1.1">
  <ows:Identifier>time</ows:Identifier>
  <Domain>2016-02-23T00:00:00.000Z/2016-02-25T00:00:00.000Z/PT8H</Domain>
  <Values>240,0,240,0,0,240</Values>
</Histogram>
```

Looking at the result we can conclude that measurements between 500.0 and 1000.0 meters are typically done during the night.

The bucket matching is setup so that each one contains its first value, but not its last value (which is contained in the next bucket instead). This is important to understand the results. Say we have a dataset with regular elevations, from 0 to 100 with a step of 10, and the request calls for elevations between 0 and 20. Then the results will look something like follows:

```
<Histogram xmlns="http://demo.geo-solutions.it/share/wmts-multidim/wmts_multi_
↪dimensional.xsd" xmlns:ows="http://www.opengis.net/ows/1.1">
  <ows:Identifier>elevation</ows:Identifier>
  <Domain>0/30/10</Domain>
  <Values>5,3,8</Values>
</Histogram>
```

That is, there values catch the intervals [0,10[, [10, 20[, and [20, 30[(to have a bucket for the images/features having elevation exactly matching 20). This will happen only if an extreme value if found, the same request filtering on elevations between 0 and 15 will return this instead:

```
<Histogram xmlns="http://demo.geo-solutions.it/share/wmts-multidim/wmts_multi_
↪dimensional.xsd" xmlns:ows="http://www.opengis.net/ows/1.1">
  <ows:Identifier>elevation</ows:Identifier>
  <Domain>0/20/10</Domain>
  <Values>5,3</Values>
</Histogram>
```

GetFeature

This operation is capable to enumerate the actual possible values combinations. The output of this operation is similar to the output of the `WFS 2.0 GetFeature` operation which is a list of features along with dimension values using the same formats as the feature info operation. This output can be used to draw the features on a map for example.

The parameters available for this operation are:

Name	Mandatory	Description
Service=WMTS	Yes	Service type identifier
Request=GetFeature	Yes	Operation name
Version=1.0.0	Yes	Standard and schema version for this operation
Layer	Yes	Layer identifier
TileMatrixSet	Yes	Tile matrix set identifier
BBOX=minx,miny,maxx,maxy	No	Bounding box corners (lower left, upper right) in CRS units
DimensionIdentifier	No	At most one per dimension, a range described as min/max, restricting the domain of this dimension
Format	Yes	The desired output format

The parameters common to the `DescribeDomains` operation work as already described above. Currently only the `application/gml+xml; version=3.1` output format is supported.

Using the same restrictions parameters we used for the second request used as an example for the `DescribeDomains` operation a `GetFeature` request will look like this:

```
http://localhost:8080/geoserver/gwc/service/wmts?REQUEST=GetFeature&Version=1.0.0&
↪Layer=some_layer&TileMatrixSet=EPSG:4326&elevation=0/500&time=2016-02-23T03:00:00.
↪000Z
```

and the result will be similar to this:

```
<?xml version="1.0" encoding="UTF-8"?><wmts:FeatureCollection xmlns:xs="http://www.w3.
↪org/2001/XMLSchema" xmlns:gml="http://www.opengis.net/gml" xmlns:wmts="http://www.
↪opengis.net/wmts/1.0">
  <wmts:feature gml:id="FID.1681">
    <wmts:footprint>
      <gml:Polygon xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:gml="http://www.
↪opengis.net/gml" xmlns:sch="http://www.ascc.net/xml/schematron" xmlns:xlink="http://
↪www.w3.org/1999/xlink" srsDimension="2" srsName="http://www.opengis.net/gml/srs/
↪epsg.xml#4326">
        <gml:exterior>
          <gml:LinearRing srsDimension="2">
            <gml:posList>-180.125 -90.125 -180.125 89.875 179.875 89.875 179.875 -90.
↪125 -180.125 -90.125</gml:posList>
          </gml:LinearRing>
        </gml:exterior>
      </gml:Polygon>
    </wmts:footprint>
    <wmts:dimension name="elevation">200.0</wmts:dimension>
    <wmts:dimension name="time">2016-02-23T03:00:00.000Z</wmts:dimension>
    <wmts:dimension name="REFERENCE_TIME">2016-02-23T00:00:00.000Z</wmts:dimension>
  </wmts:feature>
</wmts:FeatureCollection>
```

Note how this result correlate with the correspondent `DescribeDomains` operation result.

16.27 Notification community module Plugin Documentation

The notification community module is meant to be a pluggable system to listen, summarize and notify events triggered by GeoServer data and configuration manipulation to some external source, in some agreed upon format.

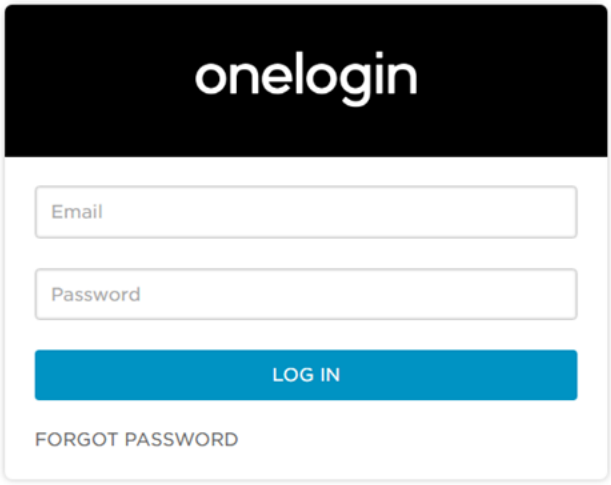
The events of interest are:

1. **Catalog configuration changes** (insert/update/removal of layers, styles, workspaces, stores, groups and so on)
2. **Data changes via WFS-T** (anything that can affect the data precise bounding box)

The system is completely pluggable in terms of notification destinations, potential targets can be direct HTTP calls to external system, message queues, log files, email. The message format can also vary depending on the target and intended usage, both in terms of contents, e.g., it could be full of details or simply an indication of what changed, and encoding, e.g., xml, json, text, html.

16.27.1 Overall architecture

The overall architecture is depicted in the following diagram:



The system basically generates a set of events, has a configuration to match them with a desired tool to send the message out (the processor). The sender can be conceived as a the combination of an "encoder" that generates the message payload and a "sender".

Each message is combined with its processor and send into a destination queue, where a thread pool picks the events and runs their processor. For some type of events, like catalog ones, the thread pool will have to be configured with just one thread to make sure the events are sent in the right order to the destinations.

16.27.2 Installing the extension

1. Download the Notification extension from the nightly GeoServer community module builds.
2. Download the Notification Common extension from the nightly GeoServer community module builds.
3. (optional) If you want use sender/encoder provided for GeoNode, download the Notification Geonode extension from the nightly GeoServer community module builds.
4. Place the JARs into the `WEB-INF/lib` directory of the GeoServer installation.

16.27.3 Usage

The usage of the extensions is based on two components that defines its behavior and logic:

- A configuration file named notifier.xml that must be present on a "notifier" subfolder of Geoserver root data directory (if extensions found no one `notifier.xml` file under notifier folder, will create a new one with default values)
- A JAR that implements the specific logic of sender/encoder

16.27.4 Configuration file

The configuration file will be parsed by XStream framework to instantiate the right classes. An example of notifier.xml have the follow content:

```
<notificationConfiguration>
<queueSize>1000</queueSize>
  <notificator>
        <messageFilter>type='Catalog'</messageFilter>
        <queueSize>1000</queueSize>
        <processorThreads>1</processorThreads>
        <genericProcessor>
          <geonodeEncoder />
          <fanoutSender>
                <username>guest</username>
                <password>guest</password>
                <host>localhost</host>
                <port>4432</port>
                <virtualHost></virtualHost>
                <exchangeName>testExchange</exchangeName>
                <routingKey>testRouting</routingKey>
          </fanoutSender>
        </genericProcessor>
  </notificator>
  <notificator>
  ...
  </notificator>
</notificationConfiguration>
```

notificationConfiguration -> **queueSize** = the size of queue that store all the notification messages.

notificationConfiguration -> **notificator** = is possible to have one or more notificator.

notificationConfiguration -> **notificator** -> **messageFilter** = the is a CQL filter, only notification message that satisfy this filter, will be processed by this notificator. Possible values are:

- `type='Catalog'`
- `type='Data'`

notificationConfiguration -> **notificator** -> **queueSize** = the size of queue to store the notification messages for specific notificator, only the notification that satisfy the CQL filter specified on `<messageFilter>` element will be pushed in this queue.

notificationConfiguration -> **notificator** -> **processorThreads** = number of threads that will be work to encode and send the notification messages. Note that for `'Catalog'` type event, this will have to be valued as 1 to make sure the events are sent in the right order to the destinations.

notificationConfiguration -> **notificator** -> **genericProcessor** = configurations for the encoder and sender components

notificationConfiguration -> **notificator** -> **geonodeEncoder** = this is a placeholder tag that must match with the alias used to map the implementation class for encoder. Based on custom implementation, additional attributes or child tags can be provided.

Note: is mandatory that one and only one implementation of encoder match with each alias.

notificationConfiguration -> **notificator** -> **fanoutSender** = this is a placeholder tag that must match with the alias used to map the implementation class for sender. Based on custom implementation, additional attributes or child tags can be provided.

Note: is mandatory that one and only one implementation of sender match with each alias.

For the case of *AMQP Fanout (RabbitMQ)* based sender implementation, the additional parameters are:

- `host` : the IP/DNS to which the underlying TCP connection is made
- `port` : port number to which the underlying TCP connection is made
- `virtualhost` (optional) : a path which acts as a namespace
- `username` (optional) : if present is used for SASL exchange
- `password` (optional) : if present is used for SASL exchange
- `exchangeName` : the name of exchange to publish the message to
- `routingKey` : identify the queue to publish the message to (ignored by fanout type)

16.27.5 Sender and encoder implementations

This plugin allow the pluggability of sender/encoder implementation, multiple implementation plugins are allowed.

The core notification extension will be resolve the implementations of the interfaces:

- `org.geoserver.notification.common.NotificationEncoder`
- `org.geoserver.notification.common.NotificationProcessor`
- `org.geoserver.notification.common.NotificationXStreamInitializer`

Based on the match between the tag names on configuration file (`notifier.xml`) and the aliases define in **NotificationXStreamInitializer**, the notification core will be use the right implementation of **NotificationEncoder** / **NotificationProcessor**.

An example of this implementation is provided by the Notification Geonode extension.

The minimal dependencies of this kind of plugin are (see `pom.xml` of *Notification Geonode* extension):

- `gs-notification-common`
- `gs-main`

The plugin must be extends/implements almost the three classes/interfaces:

NotificationXStreamDefaultInitializer: is a utility implementation of **NotificationXStreamInitializer** to define the match between **NotificationEncoder** / **NotificationProcessor** and configuration aliases:

- `getEncoderName` : this method must be return the alias for encoder
- `getSenderName` : this method must be return the alias for sender
- `getEncoderClass` : this method must be return the class for encoder
- `getSenderClass` : this method must be return the class for sender

16.28 OpenSearch for EO

16.28.1 Introduction to OpenSearch for EO

This plugin adds support for the OpenSearch for Earth Observation protocol to GeoServer. References:

- OpenSearch
- OpenSearch parameter extension
- OpenSearch Geo and Time extension
- OpenSearch for Earth Observation

The OpenSearch plugin organizes data in "Collections" and "Products":

- A collection is a set of products with some uniformity, described by some search attributes and a ISO metadata sheet
- A product is a set of images (and ancilliary information), describe by some search attributes and a O&M metadata sheet

The system allows the common EO "two level" searches, that is:

- Firsts lookup for the desired collection of data on the main OSDD document
- Once the collection is located, a second OSDD providing access to the product search is delivered

If the database contains also the OGC cross links, the Atom search outputs will also contain links allowing a client to jump from the data search to the actual data visualization and exploitation on OGC services.

Search engine storage

The OpenSearch protocol implementation relies on an extension of GeoTools `DataAccess` called `OpenSearchAccess`. At the time of writing a single implementation of such interface exists, called `JDBCOpenSearchAccess`, built and tested to work against a specific PostGIS database schema.

Note: The `JDBCOpenSearchAccess` is written in general enough terms that other databases should be usable as well, but it's likely some code improvements will be required to deal with certain databases naming restrictions (e.g., Oracle).

In the future we hope to see other implementations as well, based on storage that might be more suitable for large scale search engine such as SOLR or ElasticSearch.

16.28.2 Installing the OpenSearch for EO module

The installation of the module requires four steps:

- Setting up a PostGIS database with the required schema
- Install the OpenSearch for EO plugin and configure it
- Fill the database with information about collection and metadata

Setting up the PostGIS database

Create a PostgreSQL database and run the following SQL script:

```
https://raw.githubusercontent.com/geoserver/geoserver/master/src/community/oseo/oseo-
↪core/src/test/resources/postgis.sql
```

Downloading and installing the OpenSearch extension

The module is a community one, and thus available among the nightly builds of the desired series (the module works for 2.12.x onwards):

- Check the *download page <http://geoserver.org/download/>* for the desired series (development, stable or maintenance)
- Follow the nightly build links
- Check the `community-latest` folder
- Download the `geoserver-<VERSION>-SNAPSHOT-opensearch-eo-plugin.zip` file, and unzip its contents in the GeoServer unpacked WAR lib directory, e.g., "geoserver/WEB-INF/lib"
- Restart GeoServer

16.28.3 Configuring the OpenSearch module

The OpenSearch module needs to know upon which database perform the searches.

Follow these steps:

- Setup a standard PostGIS database pointing to the database and schema created above from the SQL file. Note down the full name of the store (e.g. `test:metadata` where `test` is the workspace and `metadata` is the store name). Besides filling the connection parameters, remember to set "expose primary keys" to true.
- Create a new store of type "JDBC based OpenSearch store" and configure the fully qualified name of the basic PostGIS store in its "store" parameter.
- Go into the "OS-EO" service configuration page and configure the "OpenSearch" store

Advanced: adding product classes

The design of the OpenSearch module is "data driven", meaning that one can materialize new search properties by just adding new columns to the product and collection tables.

In particular, both tables have a "prefix based" convention linking the properties to their respective product types, and the engine will advertise for a particular product only the properties relevant to it. For example, in an optical product, the properties starting with "opt" will be listed, but not those starting with "sar".

Here is a table of the product classes known out of the box:

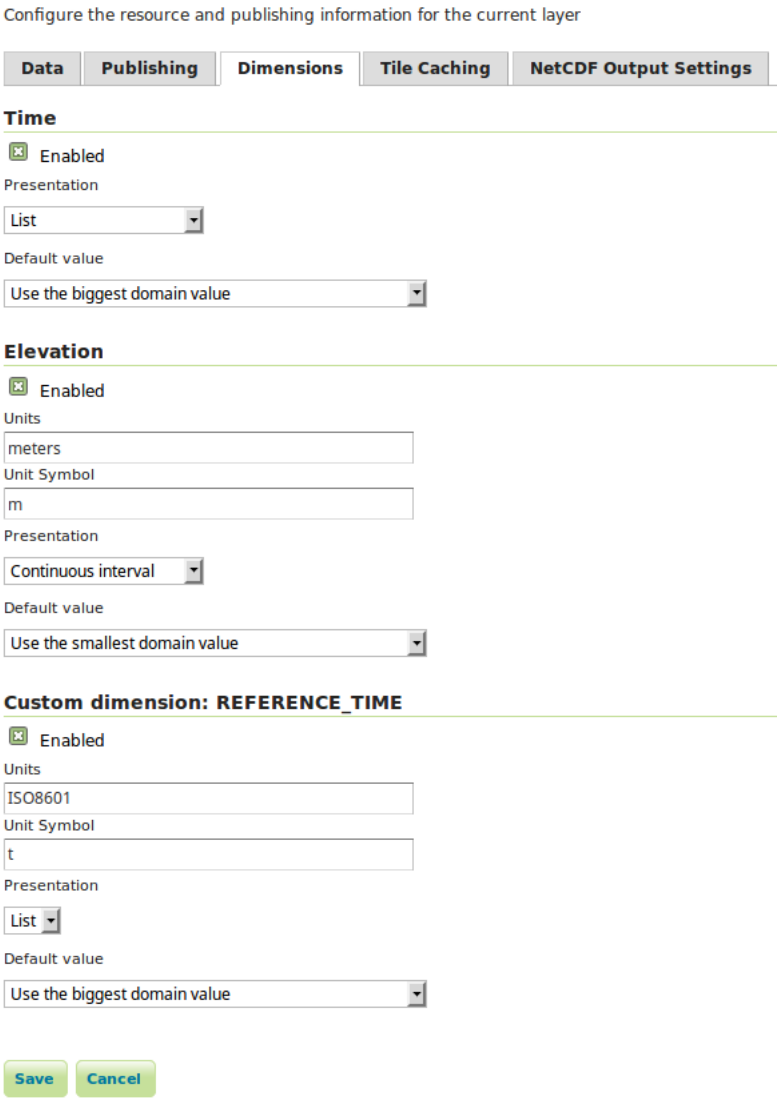


Fig. 16.20: *Configuration of a layer dimensions.*

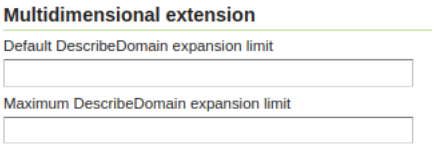
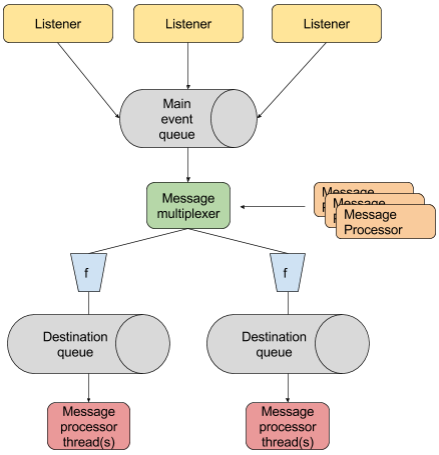


Fig. 16.21: *Configuration domain expansion limits.*



Name	Prefix	Namespace	Description
eop_generic	eop	http://www.opengis.net/eop/2.1	Base properties shared among all EO products. Never remove this class.
optical	opt	http://www.opengis.net/opt/2.1	Optical products properties
radar	sar	http://www.opengis.net/sar/2.1	Radar products properties
Altimetric	alt	http://www.opengis.net/alt/2.1	Altimetric products properties
Atmospheric	atm	http://www.opengis.net/atm/2.1	Atmospheric products properties
Limb	lmb	http://www.opengis.net/lmb/2.1	Limb products properties
ssp	ssp	http://www.opengis.net/ssp/2.1	SSP products properties

The various properties have different usages:

- The *name* is used in the collection to define the type of sensor (eoSensorType column)
- The *prefix* is used in the product table as a prefix to column name in order to associate them to a particular product type, shows up as-is in the JSON representations of the REST API, as well as the prefix in XML outputs
- The *namespace* is used in XML output, along with the prefix (e.g., `xmlns:opt="http://www.opengis.net/opt/2.1"`)

It is possible to add new product classes as well as changing the build-in ones, but care should be taken to keep product classes and database aligned. After any change to the database structure remember to "reset" the GeoServer configuration to make it re-scan the table structures.

16.28.4 The JDBC store database structure

The JDBC store uses a conventional relational structure, depicted in the following picture:

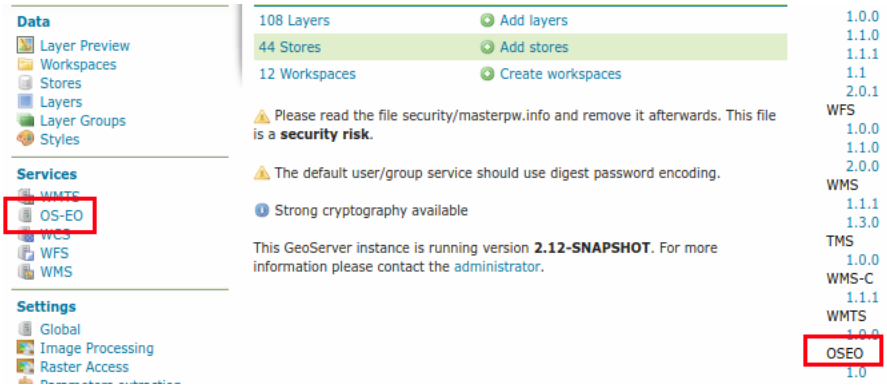


Fig. 16.22: The GeoServer home page after the OpenSearch for EO module installation.

So a `collection` has its own primary search attributes, as well as:

- A ISO metadata document as large associated text
- Zero or more OGC links pointing to where the collection is published
- Layer publishing information (for auto-generation of mosaic, layer and eventual coverage view in case the actual data resides locally)
- One or more products

A `product` in turn is associated to:

- A O&M metadata document as large associated text
- A thumbnail image, in PNG or JPEG format
- Zero or more OGC links pointing to where the product is published

The `granule` table is designed to contain per product file information in case there is a desire to publish the actual data from the same local GeoServer (but in general, OGC services might be missing or provided by a separate server).

Collections

The collection table currently looks as follows (check the SQL file in the installation instructions for a more up to date version of it):

```
create table collection (
  "id" serial primary key,
  "name" varchar,
  "primary" boolean,
  "htmlDescription" text,
  "footprint" geography(Polygon, 4326),
  "timeStart" timestamp,
  "timeEnd" timestamp,
  "productCqlFilter" varchar,
  "masked" boolean,
  "eoIdentifier" varchar unique,
```

```
  "eoProductType" varchar,
  "eoPlatform" varchar,
  "eoPlatformSerialIdentifier" varchar,
  "eoInstrument" varchar,
  "eoSensorType" varchar check ("eoSensorType" in ('OPTICAL', 'RADAR', 'ALTIMETRIC',
↪'ATMOSPHERIC', 'LIMB')),
  "eoCompositeType" varchar,
  "eoProcessingLevel" varchar,
  "eoOrbitType" varchar,
  "eoSpectralRange" varchar,
  "eoWavelength" int,
  "eoSecurityConstraints" boolean,
  "eoDissemination" varchar,
  "eoAcquisitionStation" varchar
);
```

Most of the attributes should be rather self-explanatory to those familiar with OGC Earth Observation terminology. Each attribute prefixed by "eo" is exposed as a search attribute in OpenSearch, the structure can be modified by adding extra attributes and they will show up and made searchable.

Specific attributes notes:

- A `primary` collection is normally linked to a particular satellite/sensor and contains its own products. Setting "primary" to false makes the collection "virtual" and the `productCQLFilter` field should be filled with a CQL filter that will collect all the products in the collection (warning, virtual collections are largerly untested at the moment)
- The `footprint` field is used for spatial searches, while `timeStart` and `timeEnd` are used for temporal ones
- The `htmlDescription` drives the generation of the visible part of the Atom OpenSearch response, see the dedicated section later to learn more about filling it

The `collection_metadata` table contains the ISO metadata for the given collection. The OpenSearch module has no understanding of its contents, it will simply return it as is, allowing for extra flexibility but also moving the responsibility for correctness checks completely to the author.

The `collection_ogclink` table contains the OGC links towards the services providing visualization and download access to the collection contents. See the "OGC links" section to gather more information about it.

Products

The product table currently looks as follows (check the SQL file in the installation instructions for a more up to date version of it):

```
-- the products and attributes describing them
create table product (
  "id" serial primary key,
  "htmlDescription" text,
  "footprint" geography(Polygon, 4326),
  "timeStart" timestamp,
  "timeEnd" timestamp,
  "originalPackageLocation" varchar,
  "originalPackageType" varchar,
  "thumbnailURL" varchar,
  "quicklookURL" varchar,
  "crs" varchar,
```

```
  "eoIdentifier" varchar unique,
  "eoParentIdentifier" varchar references collection("eoIdentifier") on delete␣
↪cascade,
  "eoProductionStatus" varchar,
  "eoAcquisitionType" varchar check ("eoAcquisitionType" in ('NOMINAL', 'CALIBRATION',
↪ 'OTHER')),
  "eoOrbitNumber" int,
  "eoOrbitDirection" varchar check ("eoOrbitDirection" in ('ASCENDING', 'DESCENDING
↪')),
  "eoTrack" int,
  "eoFrame" int,
  "eoSwathIdentifier" text,
  "optCloudCover" int check ("optCloudCover" between 0 and 100),
  "optSnowCover" int check ("optSnowCover" between 0 and 100),
  "eoProductQualityStatus" varchar check ("eoProductQualityStatus" in ('NOMINAL',
↪'DEGRADED')),
  "eoProductQualityDegradationStatus" varchar,
  "eoProcessorName" varchar,
  "eoProcessingCenter" varchar,
  "eoCreationDate" timestamp,
  "eoModificationDate" timestamp,
  "eoProcessingDate" timestamp,
  "eoSensorMode" varchar,
  "eoArchivingCenter" varchar,
  "eoProcessingMode" varchar,
  "eoAvailabilityTime" timestamp,
  "eoAcquisitionStation" varchar,
  "eoAcquisitionSubtype" varchar,
  "eoStartTimeFromAscendingNode" int,
  "eoCompletionTimeFromAscendingNode" int,
  "eoIlluminationAzimuthAngle" float,
  "eoIlluminationZenithAngle" float,
  "eoIlluminationElevationAngle" float,
  "sarPolarisationMode" varchar check ("sarPolarisationMode" in ('S', 'D', 'T', 'Q',
↪'UNDEFINED')),
  "sarPolarisationChannels" varchar check ("sarPolarisationChannels" in ('horizontal',
↪ 'vertical')),
  "sarAntennaLookDirection" varchar check ("sarAntennaLookDirection" in ('LEFT',
↪'RIGHT')),
  "sarMinimumIncidenceAngle" float,
  "sarMaximumIncidenceAngle" float,
  "sarDopplerFrequency" float,
  "sarIncidenceAngleVariation" float,
  "eoResolution" float
);
```

Notes on the attributes:

- The `footprint` field is used for spatial searches, while `timeStart` and `timeEnd` are used for temporal ones
- The `htmlDescription` drives the generation of the visible part of the Atom OpenSearch response, see the dedicated section later to learn more about filling it
- The `crs` attribute is optional and is used only for automatic layer publishing for collections having heterogeneous CRS products. It must contain a "EPSG:XYWZ" expression (but the product footprint still need to be expressed in WGS84, east/north oriented).
- The EO search attributes need to be filled according to the nature of the product, `eo` prefixes generic

EOP attributes, `opt` optical ones, `sar` radar ones, `atm` altimetric, `lmb` limbic, `ssp` Synthesis and Systematic Product. New attributes can be added based on the above prefixes (at the time of writing only optical and sar attributes have been tested)

The `product_metadata` table contains the O&M metadata for the given product. The OpenSearch module has no understanding of its contents, it will simply return it as is, allowing for extra flexibility but also moving the responsibility for correctness checks completely to the author.

The `product_thumb` table contains the product thumbnail, in PNG or JPEG format, for display in the OpenSearch Atom output.

The `product_ogclink` table contains the OGC links towards the services providing visualization and download access to the collection contents. See the "OGC links" section to gather more information about it.

The `htmlDescription` field

The `htmlDescription` is used to fill the user visible part of a OpenSearch ATOM response. The contents are completely freeform, but some variable can be put in the HTML that GeoServer will replace:

- `${QUICKLOOK_URL}` points to the product quicklook (at the time of writing, same as the thumbnail)
- `${THUMB_URL}` points to the product thumbnail
- `${ATOM_URL}` points to the specific Atom record at hand (either the collection or product one)
- `${OM_METADATA_URL}` points to the product O&M metadata
- `${ISO_METADATA_LINK}` points to the ISO metadata link

OGC links

The OpenSearch module implements "OGC cross linking" by adding pointers to OGC services for to collection/product visualization and download.

```
-- links for collections
create table collection_ogclink (
  "lid" serial primary key,
  "collection_id" int references collection("id") on delete cascade,
  "offering" varchar,
  "method" varchar,
  "code" varchar,
  "type" varchar,
  "href" varchar
);

-- links for products
create table product_ogclink (
  "lid" serial primary key,
  "product_id" int references product("id") on delete cascade,
  "offering" varchar,
  "method" varchar,
  "code" varchar,
  "type" varchar,
  "href" varchar
);
```

This is done by adding a set of `owc:offering` elements in the Atom response, mapping directly from the table contents:

```
<owc:offering code="http://www.opengis.net/spec/owc/1.0/req/atom/wcs">
  <owc:operation method="GET" code="GetCapabilities" href="http://localhost/sentinel2/
↪sentinel2-TCI/ows?service=WCS&amp;version=2.0.1&amp;request=GetCapabilities" type=
↪"application/xml"/>
</owc:offering>
<owc:offering code="http://www.opengis.net/spec/owc/1.0/req/atom/wmts">
  <owc:operation method="GET" code="GetCapabilities" href="http://localhost/sentinel2/
↪sentinel2-TCI/gwc/service/wmts?REQUEST=GetCapabilities" type="application/xml"/>
</owc:offering>
<owc:offering code="http://www.opengis.net/spec/owc/1.0/req/atom/wms">
  <owc:operation method="GET" code="GetCapabilities" href="http://localhost/sentinel2/
↪sentinel2-TCI/ows?service=wms&amp;version=1.3.0&amp;request=GetCapabilities" type=
↪"application/xml"/>
  <owc:operation method="GET" code="GetMap" href="http://localhost/sentinel2/
↪sentinel2:sentinel2-TCI/wms?SERVICE=WMS&amp;VERSION=1.1.1&amp;REQUEST=GetMap&amp;
↪FORMAT=image%2Fjpeg&amp;STYLES&amp;LAYERS=sentinel2%3Asentinel2-TCI&amp;SRS=EPSG
↪%3A4326&amp;WIDTH=800&amp;HEIGHT=600&amp;BBOX=-180%2C-90%2C180%2C90" type="image/
↪jpeg"/>
</owc:offering>
```

The contents of the tables need to be filled with the sane named elements of a OWC offering, the `href` one can contain a `${BASE_URL}` variable that GeoServer will replace with its own base URL.

The granule table

The granule table can be filled with information about the actual raster files making up a certain product in order to publish the collection as a GeoServer image mosaic:

```
-- the granules table (might be abstract, and we can use partitioning)
create table granule (
  "gid" serial primary key,
  "product_id" int not null references product("id") on delete cascade,
  "band" varchar,
  "location" varchar not null,
  "the_geom" geometry(Polygon, 4326) not null
);
```

The granules associated to a product can have different topologies:

- A single raster file containing all the information about the product
- Multiple raster files splitting the products spatially in regular tiles
- Multiple raster files splitting the product wavelenght wise
- A mix of the two above

Notes about the columns:

- The `band` column need to be filled only for products split in several files by bands, at the time of writing it needs to be a progressive integer starting from 1 (the module will hopefully allow more meaningful band names in the future)
- The `location` is the absolute path of the file

- The `the_geom` field is a polygon in WGS84, regardless of what the actual footprint of the file is. The polygon must represent the rectangular extend of the raster file, not its valid area (masking is to be treated separately, either with sidecar mask files or with NODATA pixels)

16.28.5 Automation with the administration REST API

The OpenSearch module supports full automation REST API that can be used to create collections, ingest products and eventually their granules. The full API is available at this URL:

- /oseo

In general terms, one would:

- Create a collection, along with description, thumbnail, metadata, OGC links
- Then create a product, along with description, thumbnail, metadata, OGC links
- Finally, and optionally, specify the granules composing the product (actually needed only if the OpenSearch subsystem is meant to be used for publishing OGC services layers too, instead of being a simple search engine.

Understanding the zip file uploads

The description of a collection and product is normally made of various components, in order to expedite data creation and reduce protocol chattines, it is possible to bulk-upload all files composing the description of collections and products as a single zip file.

Collection components

A collection.zip, sent as a PUT request to `rest/collections` would contain the following files:

Name	Optional	Description
collection.json	N	The collection attributes, matching the database structure (the prefixes are separated with a colon in this document)
description.html	Y	The HTML description for the collection
metadata.xml	Y	The metadata for the collection, in ISO format
thumbnail.png, thumbnail.jpg or thumbnail.jpeg	Y	The collection thumbnail
owsLinks.json	Y	The list of OWS links to OGC services providing access to the collection contents (typically as a time enabled layer)

Product components

A product.zip, sent as a POST request to `rest/collections/<theCollection>/products` would contain the following files:

Name	Optional	Description
product.json	N	The product attributes, matching the database structure (the prefixes are separated with a colon in this JSON document)
description.html	Y	The HTML description for the product
metadata.xml	Y	The metadata for the collection, in O&M format
thumbnail.png, thumbnail.jpg or thumbnail.jpeg	Y	The collection thumbnail
owsLinks.json	Y	The list of OWS links to OGC services providing access to the product contents (typically, a specific time slice in the collection layer, but other organizations are possible too)
granules.json	Y	The list of actual files making up the product, along with their bounding boxes, file location and eventual band name, for products splitting bands in different files. Could be a single file, a list of files split by area, or a list of files representing the various bands of a multispectral product.

It is also possible to send the zip file on the `rest/collections/<theCollection>/products/<theProduct>` resource as a PUT request, it will update an existing product by replacing the parts contained in the file. Parts missing from the file will be kept unchanged, to remove them run an explicit DELETE request on their respective REST resources.

Template variable expansion

Some of the metadata/HTML description can embed simple templating variables that GeoServer will expand while generating output. Here is a description of the variable, and where they can be used

Name	Description	Usage
${BASE_URL}	The server "base url", typically "protocol://host:port/geoserver", which can be used to save links that can easily migrate between different environments (e.g. test vs production)	OGC links, original package location download links (for products), HTML descriptions for products and collections
${ISO_METADATA_URL}	The link to a collection ISO metadata (GeoServer will point at a URL returning the metadata saved in the database)	A collection HTML description
${OM_METADATA_URL}	The link to a product O&M metadata (GeoServer will point at a URL returning the metadata saved in the database)	A product HTML description
${ATOM_URL}	The link to a collection ATOM representation, as returned by OpenSearch	A collection HTML description
${QUICKLOOK_URL}	A link to the product quicklook (GeoServer will point at a URL returning the quicklook saved in the database)	A product sample image

Usage of the API for search and integrated OGC service publishing

In this case the user intend to both use the OpenSearch module for search purposes, but also to publish actual mosaics for each collection.

In this case the following approach should is recommended:

- Create a collection via the REST API, using the ZIP file POST upload
- Create at least one product in the collection in the REST API, using the ZIP file POST upload and providing a full `granules.json` content with all the granules of said product
- Post a layer publishing description file to `/oseo/collection/{COLLECTION}/layers` to have the module setup a set of mosaic configuration files, store, layer with eventual coverage view and style

A collection can have multiple layers:

- Getting the `/oseo/collection/{COLLECTION}/layers` resource returns a list of the available ones
- `/oseo/collection/{COLLECTION}/layers/{layer}` returns the specific configuration (PUT can be used to modify it, and DELETE to remove it).
- Creation of a layer configuration can be done either by post-ing to `/oseo/collection/{COLLECTION}/layers` or by put-int to `/oseo/collection/{COLLECTION}/layers/{layer}`.

The layer configuration specification will have different contents depending on the collection structure:

- Single CRS, non band split, RGB or RGBA files, time configured as an "instant" (only `timeStart` used):

```
{
    "workspace": "gs",
    "layer": "test123",
    "separateBands": false,
    "heterogeneousCRS": false,
    "timeRanges": false
}
```

- Single CRS, multiband in single file, with a gray browse style, product time configured as a range between `timeStart` and `timeEnd`:

```
{
    "workspace": "gs",
    "layer": "test123",
    "separateBands": false,
    "browseBands": ["test123[0]"],
    "heterogeneousCRS": false,
    "timeRanges": true
}
```

- Heterogeneous CRS, multi-band split across files, with a RGB browse style ("timeRanges" not specified, implying it's handled as an instant):

```
{
    "workspace": "gs",
    "layer": "test123",
    "separateBands": true,
    "bands": [
```

```
        "VNIR",
        "QUALITY",
        "CLOUDSHADOW",
        "HAZE",
        "SNOW"
    ],
    "browseBands": [
        "VNIR[0]", "VNIR[1]", "SNOW"
    ],
    "heterogeneousCRS": true,
    "mosaicCRS": "EPSG:4326"
}
```

In terms of band naming the "bands" parameter contains coverage names as used in the "band" column of the granules table, in case a granule contains multiple bands, they can be referred by either using the full name, in which case they will be all picked, or by using zero-based indexes like `BANDNAME[INDEX]`, which allows to pick a particular band.

The same syntax is meant to be used in the `browseBands` property. In case the source is not split band, the `browseBands` can still be used to select specific bands, using the layer name as the coverage name, e.g. "test123[0]" to select the first band of the coverage.

16.29 S3 Support for GeoTiff

Support for GeoTiffs hosted on Amazon S3 or on other Amazon S3 compatible services, via a custom GeoTools GridFormat.

16.29.1 What's in the Box?

- *org.geotools.s3.geotiff*: S3 GeoTiff Format/FormatFactory/GridCoverage2dReader implementations based off of the GeoTiff versions. Only very minor changes to their parent classes.
- *org.geotools.s3.cache*: Very basic caching of images from S3 based off of EhCache.
- *S3ImageInputStreamImpl*: An implementation of ImageInputStream from JAI for reading imagery from S3. This class mainly contains the logic of stream position and chunking, while the cache package handles the actual S3 reads.

16.29.2 GeoTiffs hosted on Amazon S3

Configuration

Almost all configuration is currently done via system properties. For caching configuration, please see the class *org.geotools.s3.cache.CacheConfig*.

Usage

S3GeoTiff uses s3:// style URLs to operate. The only twist is that S3GeoTiff uses query string parameters to configure certain parameters

- *awsRegion*: Controls the region to use when connecting. Needs to be in Java enum format eg. US_WEST_2

- *useAnon*: Controls whether to authenticate anonymously. This needs to be used to connect anonymous buckets

For example:

s3://landsat-pds/L8/001/002/LC80010022016230LGN00/LC80010022016230LGN00_B1.TIF?useAnon=true&awsRegion=

Credentials

Unless *S3_USE_ANON* is set to true the [default AWS client credential chain](http://docs.aws.amazon.com/sdk-for-java/v1/developer-guide/credentials.html#using-the-default-credential-provider-chain) is used.

16.29.3 GeoTiffs hosted on other Amazon S3 compatible services

Access geotiffs on S3 servers not hosted on Amazon, e.g. https://www.minio.io/ or other. There are 2 steps to access the geofiff files. configure the server in the s3.properties file and then you can use the prefix as an alias to access the file in the S3GeoTiff module for geoserver.

Configuration

The S3 endpoints are configured in the s3.properties file. The following properties are needed for each endpoint. The prefix *alias* can be any value you choose in order to configure multiple endpoints.

- *alias.s3.endpoint*=http://your-s3-server/
- *alias.s3.user*=your-user-name
- *alias.s3.password*=your-password

Usage

Using the above configuration in the s3.properties file, the files on the S3 service can be accessed with the following URL style configuration in geoserver:

alias://bucketname/filename.tiff

- *alias*: The prefix you choose for the configuration of the endpoint
- *bucketname*: The path to the folder where the geotiffs are stored
- *filename.tif*: The name of the geotiff file

16.30 Status Monitoring

The status monitoring module add some extra information about the system in the GeoServer status page in a new tab named `Monitoring` and make that info queryable through GeoServer REST interface. This info should allow an administrator to get a quick understanding about the status of the GeoServer instance.

Library OSHI is used to retrieving system-level information without depending on native libraries or DLLs, relying solely on Apache JNA. Major operating systems (Linux, Windows and MacOX) are supported out of the box.

The available system information is:

Info	Example	Description
Operating system	Linux Mint 18	Name of the operating system and the used version
Uptime	08:34:50	Up time of the system
System average load 1 minute	0.90	System average load for the last minute
System average load 5 minutes	1.12	System average load for the last five minute
System average load 15 minute	0.68	System average load for the last fifteen minute
Number of physical CPUs	4	Number of physical CPUs / cores available
Number of logical CPUs	8	Number of logical CPUs / cores available
Number of running process	316	Total number of process running in the system
Number of running threads	1094	Total number of threads running in the system
CPU load average	4.12 %	Average load of the CPU in the last second
CPU * load	11.43 %	Load of a specific core in the last second
Used physical memory	31.58 %	Percentage of the system memory used
Total physical memory	31.4 GiB	System total memory
Free physical memory	21.4 GiB	System memory available for use
Used swap memory	0.00%	Percentage of swap memory used
Total swap memory	32.0 GiB	System total swap memory
Free swap memory	32.0 GiB	Free swap memory
File system usage	65.47 %	File system usage taking in account all partitions
Partition * used space	54.8 %	Percentage of space used in a specific partition
Partition * total space	338.9 GiB	Total space of a specific partition
Partition * free space	117.0 GiB	Free space on a specific partition
Network interfaces send	42.0 MiB	Data send through all the available network interfaces
Network interfaces received	700.4 MiB	Data received through all the available network interfaces
Network interface * send	25.0 MiB	Data send through a specific network interface
Network interface * received	250.4 MiB	Data received through a specific network interface
CPU temperature	52.00 ºC	CPU temperature
CPU voltage	1.5 V	CPU voltage
GeoServer CPU usage	3.5 %	Percentage of CPU used by GeoServer in the last second
GeoServer threads	49	Number of threads created by GeoServer
GeoServer JVM memory usage	5.83 %	Percentage of the JVM memory used by GeoServer

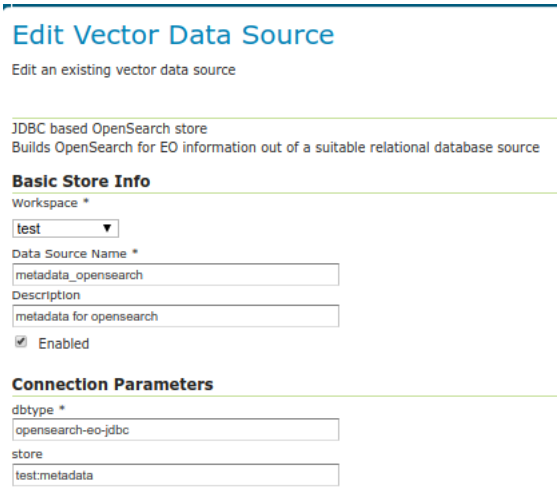
If some information is not available the special term `NOT AVAILABLE` will appear. Values will be automatically converted to best human readable unit.

16.30.1 Installing the extension

1. Download the Status Monitoring extension from the nightly GeoServer community module builds.
2. Place the JARs into the `WEB-INF/lib` directory of the GeoServer installation.

16.30.2 Usage

The system information will be available in the GeoServer status page in the `Monitoring` tab (the following image only shows part of the available system information):



If the `Monitoring` tab is not present, it means that the plugin was not installed correctly. The `Monitoring` tab content will be refreshed automatically every second.

16.30.3 REST interface

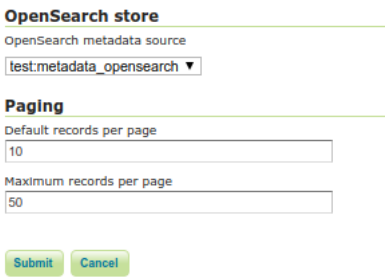
It is possible to request the available system information (monitoring data) through GeoServer REST API. The supported formats are XML, JSON and HTML.

The available REST endpoints are:

```
/geoserver/rest/about/monitoring

/geoserver/rest/about/monitoring.json

/geoserver/rest/about/monitoring.xml

/geoserver/rest/about/monitoring.html
```

The HTML representation of the system data is equal to the `Monitoring` tab representation:

The XML and JSON representations are quite similar, for each system information the following attributes will be available:



Name	Description
name	name of the metric
available	TRUE if the system information value is available
description	description of this system information
unit	unit of the system information, can be empty
category	category of this system information
priority	this value can be used to render the metrics in a predefined order
identifier	identifies the resource associated with the metric, e.g. file partition name

Example of XML representation:

```
    <metrics>
<metric>
  <value>99614720</value>
  <available>true</available>
  <description>Partition [/dev/nvme0n1p2] total space</description>
  <name>PARTITION_TOTAL</name>
  <unit>bytes</unit>
  <category>FILE_SYSTEM</category>
  <identifier>/dev/nvme0n1p2</identifier>
  <priority>507</priority>
</metric>
            (...)
```

Example of JSON representation:

```
{
    "metrics": {
        "metric": [
            {
          "available": true,
          "category": "FILE_SYSTEM",
          "description": "Partition [/dev/nvme0n1p2] total space",
          "identifier": "/dev/nvme0n1p2",
          "name": "PARTITION_TOTAL",
          "priority": 507,
          "unit": "bytes",
          "value": 99614720
      },
            (...)
```

16.31 NSG Profile

NSG Profile introduces a new operation for WFS 2.0.2 named PageResults. This operation will allow clients to access paginated results using random positions.

The current WFS 2.0.2 OGC specification defines a basic pagination support that can been used to navigate through features responses results.

Pagination is activated when parameters count and startIndex are used in the query, for example:

```
http://<host>/geoserver/ows?service=WFS&version=2.0.0&request=GetFeature&
↪typeNames=topp%3Atasmania_roads&count=5&startIndex=0
```

In this case each page will contain five features. The returned feature collection will have the next and previous attributes which will contain an URL that will allow clients to navigate through the results pages, i.e. previous page and next page:

```
<wfs:FeatureCollection
   previous="http://localhost:8080/geoserver/wfs?
   REQUEST=GetFeature&
   VERSION=2.0.0&
   TYPENAMES=topp:tasmania_roads&
   SERVICE=WFS&
   COUNT=2&
   STARTINDEX=0"
   next="http://localhost:8080/geoserver/wfs?
   REQUEST=GetFeature&
   VERSION=2.0.0&
   TYPENAMES=topp:tasmania_roads&
   SERVICE=WFS&
   COUNT=2&
   STARTINDEX=4"
   numberMatched="14"
   numberReturned="2">
```

This means that this type of navigation will always be sequential, if the client is showing page two and the user wants to see page five the client will have to:

1. request page three and use the provided next URL to retrieve page four
2. request page four and use the provided next URI to retrieve page five

This is not an ideal solution to access random pages, which is common action. PageResults operation adds a standard way to perform request random pages directly.

16.31.1 Installing the extension

1. Download the NSG Profile extension from the nightly GeoServer community module builds.
2. Place the JARs into the `WEB-INF/lib` directory of the GeoServer installation.

16.31.2 Configure the extension

The root directory inside the GeoServer data directory for the nsg-profile community module is named nsg-profile and all the configurations properties are stored in a file named **configuration.properties**.

All configuration properties are changeable at runtime, which means that if a property is updated the module take it into account.

When the application starts if no configuration file exists one with the default values is created.

The GetFeature requests representations associated with an index result type is serialized and stored in the file system in a location that is configurable.

The default location, relative to the GeoServer data directory, is nsg-profie/resultSets.

The GetFeature request to resultSetID mapping is stored by default in an H2 DB in nsg-profie/resultSets folder; for details on database configuration see GeoTools JDBCDataStore syntax

The configuration properties are the follows:

Name	Default Value	Description
resultSets.storage.path	${GEOSERVER_DATA_DIR}/nsg-profile/resultSets	Path where to store GetFeature requests representations
resultSets.timeToLive	600	How long a GetFeature request should be maintained by the server (in seconds)
resultSets.db.dbtype	h2	DB type used to store GetFeature request to resultSetID mapping
resultSets.db.database	${GEOSERVER_DATA_DIR}/nsg-profile/db/resultSets	path where to store GetFeature request to resultSetID mapping
resultSets.db.user	sa	database user username
resultSets.db.password	sa	database user password
resultSets.db.port		database port to connect to
resultSets.db.schema		database schema
resultSets.db.host		server to connect to

16.31.3 Index Result Type

The **index result type** extends the WFS **hits result type** by adding an extra attribute named **resultSetID** to the response. The **resultSetID** attribute can then be used by the **PageResults operation** to navigate randomly through the results.

A GetFeature request that uses the index result type should look like this:

```
http://<host>/geoserver/ows?service=WFS&version=2.0.0&request=GetFeature&
↪typeNames=topp%3Atasmania_roads&resultType=index
```

The response of a GetFeature operation when the index result type is used should look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<wfs:FeatureCollection
 numberMatched="14"
 numberReturned="0"
 resultSetID="ef35292477a011e7b5a5be2e44b06b34"
 xmlns:fes="http://www.opengis.net/fes/2.0"
 xmlns:gml="http://www.opengis.net/gml/3.2"
 xmlns:ows="http://www.opengis.net/ows/1.1"
 xmlns:wfs="http://www.opengis.net/wfs/2.0"
 xmlns:xlink="http://www.w3.org/1999/xlink"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.opengis.net/wfs/2.0
 http://schemas.opengis.net/wfs/2.0/wfs.xsd"/>
```

The **resultSetID** is an unique identifier that identifies the original request.

Clients will use the **resultSetID** with the PageResults operation to reference the original request.

If pagination is used, the previous and next attributes should appear as in hits result type request.

16.31.4 PageResults Operation

The **PageResults operation** allows clients to query random positions of an existing result set (stored GetFeature request) that was previously created using the **index result type** request.

The available parameters are this ones:

Name	Mandatory	Default Value
service	YES	WFS
version	YES	2.0.2
request	YES	PageResults
resultSetID	YES	
startIndex	NO	0
count	NO	10
outputFormat	NO	application/gml+xml; version=3.2
resultType	NO	results
timeout	NO	300

The two parameters that are not already supported by the GetFeature operation are the **resultSetID** parameter and the **timeout** parameter.

1. The **resultSetID** parameter should reference an existing result set (stored GetFeature request).

   A typical PageResults request will look like this:

   ```
   http://<host>/geoserver/ows?service=WFS&version=2.0.2&request=PageResults&
   ↪resultSetID=ef35292477a011e7b5a5be2e44b06b34&startIndex=5&count=10&
   ↪outputFormat=application/gml+xml; version=3.2&resultType=results
   ```

   This looks like a GetFeature request where the **query expression was substituted by the resultSetID parameter**.

2. The **timeout** parameter is not implemented yet.

The following parameters of index request are override using the ones provided with the PageResults operation or the default values:

1. startIndex
2. count
3. outputFormat
4. resultType

and finally the GetFeature response is returned.

16.32 GHRSST NetCDF output

GHRSST is Group for High Resolution Sea Surface Temperature. Among its various activities it issued a specification on how sea surface temperature data should be organized in NetCDF files.

The NetCDF GHRSST module allows to generate complaint GHRSST files as WCS outputs, given a compliant GHRSST input.

16.32.1 Installation

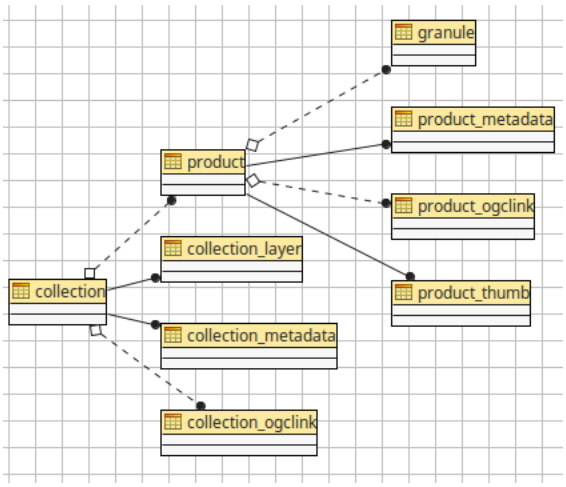
As a community module, the package needs to be downloaded from the nightly builds, picking the community folder of the corresponding GeoServer series (e.g. if working on GeoServer master nightly builds, pick the zip file form `master/community-latest`).

To install the module, unpack the zip file contents into GeoServer own `WEB-INF/lib` directory and restart GeoServer.

For the module to work, the *NetCDF* and *NetCDF Output Format* extensions must also be installed.

16.32.2 Input preparation

A GHRSST file contains multiple variables that are related with each other, and should be explored toghether in order to better understand the data. Thus, it is assumed that the source GHRSST file is published as a single coverage view holding all the variables as bands, retaining their native name (this is important for the plugin to work):



A GHRSST output must also have a time, so the time dimension of this layer should be enabled (the output generation will fail with an error otherwise).

At the time of writing a coverage view requires the source bands to be of uniform data type, and the data sources might not be. In case setting up the view is not possible with the data available, a NCML file can be used to reprocess the source NetCDF into one that has bands with uniform data type. A downloadable example has been provided to facilitate setting up the view.

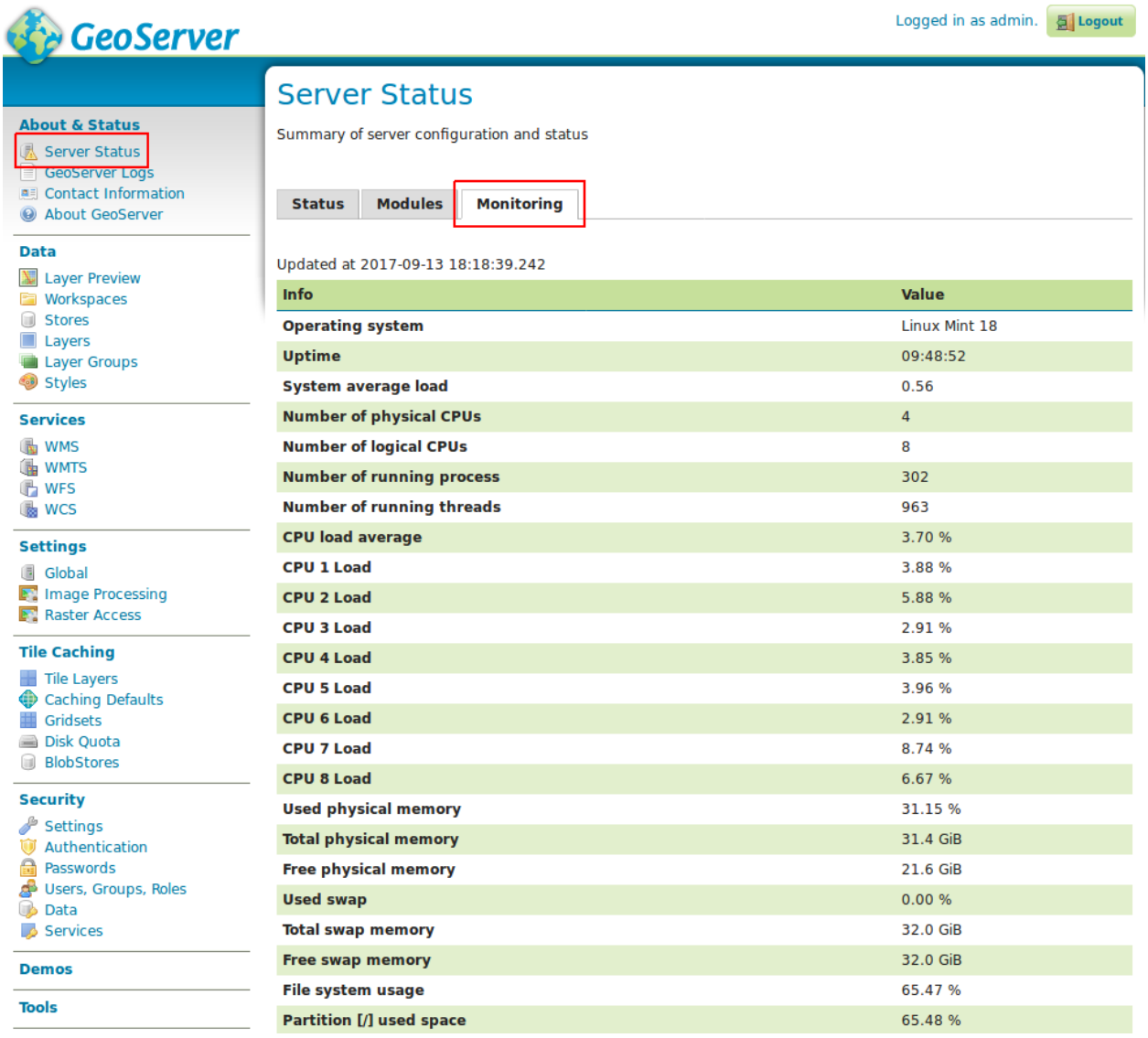
`Download the reference NCML transformation`

The GHRSST may also have to be setup in a image mosaic in order to provide a deep temporal layer that users can select data from. The image mosaic setup can be complex, so a downloadable example has been provided for it as well (will require some changes, at a minimum, fix the paths at the bottom of indexer.xml, and the database connection parameters in the two datastore properties files).

`Download the sample mosaic configuration files`

16.32.3 Configuring GHRSST output

The normal WCS NetCDF output will pick the first band of a coverage and generate a single variable NetCDF output. When the GHRSST plugin is installed, a new UI element will show up that enables GHRSST output:



Notes about the configuration UI:

- Various normal configurations such as variable name, unit of measure, and data packing will be ignored (each variable in GHRSST has its own assigned data type and packing, as from specification)
- For the output to be compliant, enable copy of both global and per variable attributes
- The RDAC, Processing Level, SST Type and Product String have to be filled in order to generate a valid GHRSST file name in output. The user interface provides auto-complete with names picked from the specification, but others can be inputed as well.

For the output to generate correctly the coverage band names have to follow exactly the expected specification variable names (which comes naturally if the input is valid GHRSST), variable will be re-packed in

output according to specification, so even if the inputs are all floats, the output will follow the expected data types.

Any extra coverage band not present in the specification will be copied from input to output un-modified.

16.33 Monitoring Hibernate storage

The monitor hibernate storage allows to track the requests made against a GeoServer instance in a relational database, as opposed to keeping the data in memory for a short time, or logging it on a audit file.

16.33.1 Installing the Hibernate Monitor Extension

Note: If performing an upgrade of the monitor extension please see *Upgrading*.

As a community module, the package needs to be downloaded from the nightly builds, picking the community folder of the corresponding GeoServer series (e.g. if working on GeoServer master nightly builds, pick the zip file form `master/community-latest`).

To install the module, unpack the zip file contents into GeoServer own `WEB-INF/lib` directory and restart GeoServer.

For the module to work, the *Monitoring* extensions must also be installed.

16.33.2 Hibernate storage Configuration

Many aspects of the monitor extension are configurable. The configuration files are stored in the data directory under the `monitoring` directory:

```
<data_directory>
    monitoring/
        db.properties
        hibernate.properties
```

In particular: * **db.properties** - Database configuration when using database persistence. * **hibernate.properties** - Hibernate configuration when using database persistence.

Monitor Storage

How request data is persisted is configurable via the `storage` property defined in the `monitor.properties` file. The following values are supported for the `storage` property:

- **memory** - Request data is to be persisted in memory alone.
- **hibernate** - Request data is to be persisted in a relational database via Hibernate.

The default value is `memory`, in order to use hibernate the `storage` configuration needs to be switced to `hibernate`.

16.33.3 Database Persistence

The monitor extension is capable of persisting request data to a database via the Hibernate library.

Note: In order to utilize hibernate persistence the hibernate extension must be installed on top of the core monitoring extension. See the *Installing the Monitor Extension* for details.

Configuration

General

In order to activate hibernate persistence the `storage` parameter must be set to the value "hibernate":

```
storage=hibernate
```

The hibernate storage backend supports both the `history` and `live` modes however care should be taken when enabling the `live` mode as it results in many transactions with the database over the life of a request. Unless updating the database in real time is required the `history` mode is recommended.

Database

The file `db.properties` in the `<GEOSERVER_DATA_DIR>/monitoring` directory specifies the Hibernate database. By default an embedded H2 database located in the `monitoring` directory is used. This can be changed by editing the `db.properties` file:

```
# default configuration is for h2
driver=org.h2.Driver
url=jdbc:h2:file:${GEOSERVER_DATA_DIR}/monitoring/monitoring
```

For example to store request data in an external PostgreSQL database, set `db.properties` to:

```
driver=org.postgresql.Driver
url=jdbc:postgresql://192.168.1.124:5432/monitoring
username=bob
password=foobar
defaultAutoCommit=false
```

In addition to `db.properties` file is the `hibernate.properties` file that contains configuration for Hibernate itself. An important parameter of this file is the hibernate dialect that informs hibernate of the type of database it is talking to.

When changing the type of database both the `databasePlatform` and `database` parameters must be updated. For example to switch to PostgreSQL:

```
# hibernate dialect
databasePlatform=org.hibernate.dialect.PostgreSQLDialect
database=POSTGRESQL

# other hibernate configuration
hibernate.use_sql_comments=true
generateDdl=true
hibernate.format_sql=true
showSql=false
```

```
hibernate.generate_statistics=true
hibernate.session_factory_name=SessionFactory
hibernate.hbm2ddl.auto=update
hibernate.bytecode.use_reflection_optimizer=true
hibernate.show_sql=false
```

Hibernate

As mentioned in the previous section the `hibernate.properties` file contains the configuration for Hibernate itself. Aside from the database dialect parameters it is not recommended that you change this file unless you are an experienced Hibernate user.

16.33.4 Upgrading

The monitoring extension uses Hibernate to persist request data. Changes to the extension over time affect the structure of the underlying database, which should be taken into consideration before performing an upgrade. Depending on the nature of changes in an upgrade, it may involve manually making changes to the underlying database before deploying a new version of the extension.

The sections below provides a history of such changes, and recommended actions that should be taken as part of the upgrade. Upgrades are grouped into two categories:

- **minor** upgrades that occur during a minor GeoServer version change, for example going from 2.1.2 to 2.1.3. These changes are backward compatible in that no action is specifically required but potentially recommended. In these cases performing an upgrade without any action still result in the monitoring extension continuing to function.
- **major** upgrades that occur during a major GeoServer version change, for example going from 2.1.2 to 2.2.0. These changes *may* be backward compatible, but not necessarily. In these cases performing an upgrade without any action could potentially result in the monitoring extension ceasing to function, and may result in significant changes to the underlying database.

For each change the following information is maintained:

- The released version containing the change
- The date of the change
- The subversion revision of the change
- The jira issue referring to the change

The date and subversion revision are especially useful if a nightly build of the extension is being used.

Minor upgrades

Column `resource` renamed to `name` in `request_resources` table

- *Version*: n/a, extension still community status
- *Date*: Dec 09, 2011
- *Subversion revision*: 16632
- *Reference*: GEOS-4871

Upgrading without performing any action will result in the `name` column being added to the `request_resources` table, leaving the `resource` column intact. From that point forward the `resource` column will essentially be ignored. However no data from the `resource` column will be migrated, which will throw off reports, resource access statistics, etc... If you wish to migrate the data perform one of the following actions two actions.

The first is a *pre* upgrade action that involves simply renaming the column before deploying the new monitoring extension:

```
ALTER TABLE request_resources RENAME COLUMN resource to name;
```

Alternatively the migration may occur *post* upgrade:

```
UPDATE TABLE request_resources SET name = resource where name is NULL;
ALTER TABLE request_resources DROP COLUMN resource;
```

Column `remote_user_agent` added to `request` table

- *Version*: n/a, extension still community status
- *Date*: Dec 09, 2011
- *Subversion revision*: 16634
- *Reference*: GEOS-4872

No action should be required here as Hibernate will simply append the new column to the table. If for some reason this does not happen the column can be added manually:

```
ALTER TABLE request ADD COLUMN remote_user_agent VARCHAR(1024);
```

Major upgrades

16.34 Geoserver Task Manager

The GeoServer Task Manager is a tool that allows scheduled tasks and batches to run inside a GeoServer instance. The initial purpose of the task manager was to synchronize data and metadata between different geoserver instances and their data sources, as to automise and assist a particular workflow within an organisation that includes pre-tested publications and frequent updates. However, the task manager offers an extensive and flexible API that makes it possible to write any kind of extensions and customisations, allowing other purposes as well.

16.34.1 Basic Concepts

The two main kinds of objects of the Task Manager are *configurations* and *batches*. Task Manager also allows the creation of *Templates* for configurations.

Configurations

The configuration is the central object in the Task Manager. A configuration is typically linked to a data object, such as a GeoServer layer, and serves as an entry point to the tasks and batches related to this data object.

A configuration has a unique name, a description and a workspace. It contains three groups of objects:

- `Attributes`: The attributes contain information about this configuration that can be shared between the different tasks of this configuration. An attribute has a name and a value. Each attribute is associated with at least one task parameter (see below). Attributes inherit their validation properties from their associated parameters, such as its accepted values and whether it is required.
- `Tasks`: Each task configures an operation that can be executed on this configuration. Each task has a name that is unique within the configuration, a type and a list of parameters with each a name and a value. The full name of a task is donated as *configuration-name/task-name* (which serves as a unique identifier for the task). The task's type is chosen from a list of available task types which define different kinds of operations (for example: copy a database table, publish a layer, ..) and expects a list of parameters that each has a name and a type. A parameter may or may not be required. The parameter type defines the accepted values of the parameter. Parameter types are dependent types when the list of accepted values depends on the value of another parameter (for example: tables inside a database). A parameter value is either a literal or a reference to an attribute of the form `${attribute-name}`.
- `Batches`.

Batches

A batch is made of an ordered sequence of tasks that can either be run on demand or be scheduled to run repeatedly at specific times. There are two kinds of batches:

- `Configuration batches`: these are batches that belong to a configuration. All of the tasks inside this batch are tasks that belong to that same configuration.
- `Independent batches`: these are batches that do not belong to a configuration. They may contains tasks from any existing configuration.

A batch has a name, a description and a workspace. The name of a batch must be unique amongst its configuration or amongst all independent batches. The full name of a batch is denoted as *[configuration-name:]batch-name* which serves as a unique identifier for the batch.

Configuration batches that have a name starting with a `@`, are hidden from the general batch overview and are only accessible from their configuration. Hidden batch names may be reserved for special functions. At this point, there is only one such case (see *Initializing templates*).

A batch can be run manually if the following conditions are met:

- the list of tasks is non-empty;
- the operating user has the security rights to do so (see Security).

A batch will be run automatically on its scheduled time if the following conditions are met:

- the list of tasks is non-empty;
- the batch is enabled;
- the batch has a frequency configured other than `NEVER`;
- the batch is independent or its configuration has been completed, i.e. validated without errors (in some cases a configuration may be saved before it is validated, see *Initializing templates*).

Running a batch

The batch is executed in two phases:

- RUN phase: tasks are executed in the defined order. If an error occurs or the run is manually intermitted, cease execution and go to ROLLBACK phase. If all tasks finish succesfully, go to COMMIT phase.
- COMMIT/ROLLBACK phase: tasks are committed or rollbacked in the *opposite* order.

Consider a batch with three tasks

B = T1 -> T2 -> T3.

A normal run would then be

run T1 -> run T2 -> run T3 -> commit T3 -> commit T2 -> commit T1.

However, if T2 fails, the run would be

run T1 -> run T2 (failure) -> rollback T1.

Most tasks support COMMIT/ROLLBACK by creating temporary objects that only become definite objects after a COMMIT. The ROLLBACK phase then simply cleans up those temporary objects. However, some particular task types may not support the COMMIT/ROLLBACK mechanism (in which case running them is definite).

The commit phase happens in opposite order because dependencies in the old version of the data often requires this. A concrete example may clear things up. Imagine that *T1* copies a database table *R* from one database to another, while *T2* creates a view *V* based on that table, so *V* depends on *R*. If the table and view already exist in older versions (*R_old* and *V_old*), they must not be removed until the COMMIT phase, so that their original state remains in the case of a ROLLBACK. During the COMMIT phase, *R_old* and *V_old* are removed, but it is not possible to remove *R_old* until *V_old* is removed. Therefore it is necessary to commit *T2* before *T1*.

The COMMIT phase typically replaces old objects with the new objects that have a temporary name. Since tasks often create objects that depend on objects of the previous tasks, these objects contain references to temporary names. Which means that when the temporary object is committed and becomes the real object, references in depending objects must also be updated. For this purpose, a tasks that uses a temporary object from a previous task registers a *dependency*, which is essentially an update added to the commit phase of that previous task.

If *T3* has a dependency on task *T1* that we call *D1*, the following happens:

run T1 -> run T2 -> run T3, register D1 -> commit T3 -> commit T2 -> commit T1, update D1.

Let's make it clearer again using an example. During the RUN phase *T1* creates table *R1_temp* and *T2* creates *V1_temp* that depends on *R1_temp*, this dependency will be registered. During the commit phase, *T2* will replace *V1* by *V1_temp*. Then, *T1* will replace *T1* by *T1_temp*. However, *V1* may still reference *T1_temp* which no longer exists. Therefore, *T1* will use the registered dependency to update *V1* to refer to *T1* instead of *T1_temp*.

Within a batch run, each task that has yet started has a status. These are the possible statuses:

- RUNNING: the task is currently running.
- WAITING_TO_COMMIT: the task has finished running, but is waiting to commit (or rollback) while other tasks are running or committing (or rolling back).
- COMMITTING: the task is currently committing.
- ROLLING_BACK: the task is currently rolling back.
- COMMITTED: the task was succesfully committed.
- ROLLED_BACK: the task was succesfully rolled back.
- NOT_COMMITTED: the task was supposed to commit but failed during the commit phase.
- NOT_ROLLED_BACK: the task was supposed to roll back but failed during roll back phase.

A task is consired finished if its status is not `RUNNING`, `WAITING_TO_COMMIT`, `ROLLING_BACK` or `COMMITTING`. A batch run does not have its own status, but it takes on the status of the last task that has started but is not `COMMITTED` or `ROLLED_BACK`. A batch run is considered finished if its status is not `RUNNING`, `WAITING_TO_COMMIT` or `COMMITTING`.

There is concurrency protection both on the level of tasks and batches. A single batch can never run simultaneously in multiple runs (the second run will wait for the first one to finish). A single task can never run simultaneously in multiple runs, even if part of a different batch. A single task can also not commit simultaneously in multiple runs.

Templates

Templates are in every way identical to configurations, with the exception of:

- they are never validated when saved (their attributes need not be filled in) and
- their tasks and batches can never be executed.

A template is used as a blueprint for the creation of configurations that are very similar to each other. Typically, the tasks are all the same but the attribute values are different. However, a template may also have attribute values filled in that serve as defaults.

Once a configuration is created from a template, it is independent from that template (changes to the template do not affect it). The configuration can then be modified like any other configuration, including the removal, addition and manipulation of tasks.

Initializing templates

An initializing template is any template that has a batch named `@Initialize` (case sensitive), which configures special behaviour. The purpose of this batch is to execute some tasks that must have been done at least once until some other tasks can actually be configured. For example, you may want to create a vector layer based on that table copied from a source database, then synchronise this layer to a target geoserver. The task that synchronizes a layer to the external geoserver will expect an existing configured layer, which you cannot create until you have copied the table first. The `@Initialize` batch would in this case copy the table from the source and create a layer in the local geoserver.

When creating a configuration from this template, configuration happens in two phases

- 1. Initially, only attributes related to tasks in the `@Initialize` batch must be configured. When the configuration is saved, the `@Initialize` batch is automatically executed.
- 2. Now, all other attributes and tasks must be configured and the configuration must be saved again.

This is the only case that a configuration can be saved before all the required attributes are filled in. Mind that batches will not be scheduled or visible in the general overview until the batch has been saved again (and the attributes have thus been validated).

16.34.2 User's Guide

- *Installation*
- *Server Configuration*
- *Security*
- *Graphical User Interface*

- *Task Types*
- *Import Tool*
- *Examples*

Installation

To install the GeoServer Task Manager extension:

- Download the extension from the GeoServer Download Page. The file name is called `geoserver-*-taskmanager-core-plugin.zip`, where `*` is the version/snapshot name. For *S3 support*, also install the plugin with name `geoserver-*-taskmanager-s3-plugin.zip`, where `*` is the version/snapshot name.
- Extract this file and place the JARs in `WEB-INF/lib`.
- Perform any configuration required by your servlet container, and then restart. On startup, Task Manager will create a configuration directory `taskmanager` in the GeoServer Data Directory. You will be able to see the Task Manager configuration pages from the GeoServer WebGUI menu.

Server Configuration

Configuration Database & Clustering

By default, Task Manager will create a H2 database in its configuration directory. This can be easily changed to any JDBC resource via the `taskmanager.properties` file.

The configuration directory also contains a Spring configuration file called `taskManager-applicationContext.xml` which allows more advanced configuration.

TaskManager uses Quartz Scheduler. If you are running Task Manager in a clustered environment, you must configure Quartz to use a database as well as Task Manager. See the commented block in the Spring configuration and the Quartz documentation for further instructions. The database used by Quartz may be the same as the Task Manager configuration database.

Furthermore, a property should be added to the `taskmanager.properties` file each of the nodes except for one: `batchJobService.init=false`. This is necessary because otherwise all of the nodes will attempt to load all of the same batches in to the clustered quartz database at the same time at start-up, which is likely to cause issues. This initialisation needs to happen only once for the entire cluster.

Databases

Task Manager allows any number of databases to be used both as sources and targets for data transfer operations. These are configured via the Spring configuration file. Currently only PostGIS is fully supported, either via JNDI or directly via JDBC.

```
<bean class="org.geoserver.taskmanager.external.impl.PostgisDbSourceImpl">
    <property name="name" value="mypostgisdb"/>
    <property name="host" value="hostname" />
    <property name="db" value="dbname" />
    <!-- optional --> <property name="schema" value="schema" />
    <property name="username" value="username" />
    <property name="password" value="password" />
    <!-- optional, for security purposes -->
    <property name="roles">
```

```
      <list>
       <value>ROLE1</value>
       <value>ROLE2</value>
      </list>
    </property>
</bean>
```

```
<bean class="org.geoserver.taskmanager.external.impl.PostgisJndiDbSourceImpl">
    <property name="name" value="mypostgisjndidb" />
    <property name="jndiName" value="java:/comp/env/jdbc/my-jndi-source" />
    <!-- optional --> <property name="schema" value="schema" />
    <!-- optional, if database has different jndi name on target geoserver servers -->
     <property name="targetJndiNames">
     <map>
        <entry key="mygs" value="java:/comp/env/jdbc/my-jndi-source-on-mygs" />
     </map>
    </property>
    <!-- optional, for security purposes -->
    <property name="roles">
      <list>
       <value>ROLE1</value>
       <value>ROLE2</value>
      </list>
    </property>
</bean>
```

Roles can be specified for *security<#security>* purposes.

There is also support for Informix, but it only works as a source database (not for publishing).

```
<bean class="org.geoserver.taskmanager.external.impl.InformixDbSourceImpl">
    <property name="driver" value="com.informix.jdbc.IfxDriver"/>
    <property name="connectionUrl" value="jdbc:informix-sqli://informix-server:1539" /
↪>
    <property name="username" value="username" />
    <property name="password" value="password" />
</bean>
```

External GeoServers

Task Manager allows any number of external geoservers to be used as targets for layer publications. These are configured via the Spring configuration file.

```
<bean class="org.geoserver.taskmanager.external.impl.ExternalGSImpl">
    <property name="name" value="mygs"/>
    <property name="url" value="http://my.geoserver/geoserver" />
    <property name="username" value="admin" />
    <property name="password" value="geoserver" />
</bean>
```

File Services

File Services are used to upload and access files such as raster layers. They are configured via the Spring configuration file.

Regular File Service

Regular file services provide support for rasters that are stored on the hard drive.

```
<bean class="org.geoserver.taskmanager.fileservice.impl.FileServiceImpl">
    <property name="rootFolder" value="/tmp"/>
    <property name="name" value="Temporary Directory"/>
    <property name="roles">
      <list>
       <value>ROLE1</value>
       <value>ROLE2</value>
      </list>
    </property>
</bean>
```

Roles can be specified for *security<#security>* purposes.

S3 File Service

S3 File Services provide support for rasters that are stored on an S3 compatible server.

They do not need to be configured via the application context, but are taken from the properties file provided via the property `s3.properties.location` (see S3 DataStore).

A service will be created for each service and each bucket. We must add one line per alias to the `s3.properties` file:

```
alias.s3.rootfolder=comma,separated,list,of,buckets
```

The above example will create five s3 file services: alias-comma, alias-separated, alias-list, alias-of and alias-buckets.

Roles can optionally be specified for *security<#security>* purposes as follows:

```
alias.s3.rootfolder.bucket=comma,separated,list,of,roles
```

Prepare script

The task manager GUI allows immediate upload of files to file services for local publication. It may be handy to perform some preprocessing tasks on the uploaded data before publication (such as GDAL commands). You may do this by creating a file in the taskmanager configuration directory named *prepare.sh*. If the user ticks the prepare checkbox in the upload dialog, this script will be run with the uploaded file as its first parameter.

Security

Each configuration and each independent batch is associated with a workspace in GeoServer (when the workspace field is empty, it is automatically associated with the default workspace in geoserver). The configuration or batch takes its security permissions directly from this workspace.

- If the user has reading permissions on the workspace, they may view the configuration or batch.
- If the user has writing permissions on the workspace, they may run the batch or the batches in the configuration.
- If the user has administrative permissions on the workspace, they may edit the configuration/batch.

Each Database or File Service may be associated with a list of roles. If you do so, only users with those roles will have access to the database or file service in question. If you want to disable security restrictions, do not include the `roles` property at all (because an empty list will result in no access.)

Graphical User Interface

Currently GeoServer Task Manager can only be configured and operated from the GeoServer WebGUI.

Templates

From the templates page, new templates can be created (or copied from existing templates), existing templates can be edited and removed.

Info	Value
Operating system	Linux Mint 18
Uptime	07:40:58
System average load	1.01
Number of physical CPUs	4
Number of logical CPUs	8
Number of running process	302
Number of running threads	1026
CPU load average	7.69 %
CPU 1 Load	8.54 %
CPU 2 Load	8.37 %
CPU 3 Load	8.01 %
CPU 4 Load	8.55 %
CPU 5 Load	7.48 %
CPU 6 Load	6.85 %
CPU 7 Load	6.71 %
CPU 8 Load	6.98 %
Used physical memory	24.11 %
Total physical memory	31.4 GiB
Free physical memory	23.8 GiB
Used swap	0.00 %
Total swap memory	32.0 GiB
Free swap memory	32.0 GiB
File system usage	65.46 %
Partition [/] used space	65.47 %
Partition [/] total space	338.9 GiB
Partition [/] free space	117.0 GiB
Partition [/dev/nvme0n1p2] used space	30.02 %

Once you open a new or existing template, attributes, tasks and batches can be edited. The attribute table adjusts automatically based on the information in the tasks table; and only the values must be filled in. In the task table, the name and parameters of each task can be edited, and new tasks can be created. Batches can be created and edited from here as well, however the template must exist in order to be able to do that (in case of a new template, you must click `apply` once before you can create new batches). New tasks must also be saved (again, via the `apply` button) before they can be added to a batch.

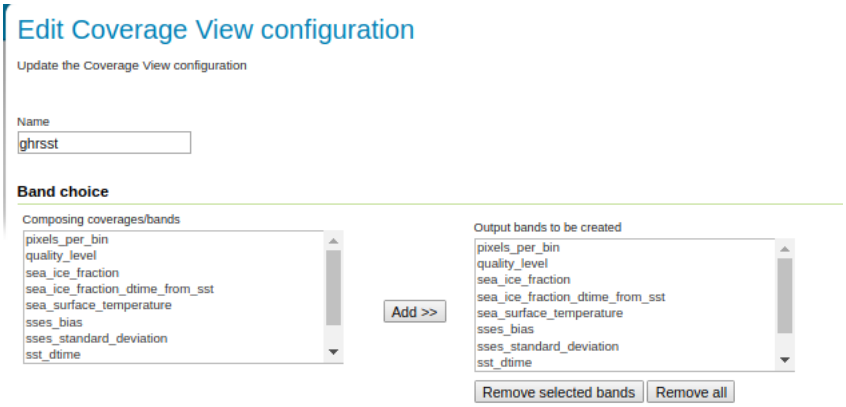


Fig. 16.23: *Setting up a coverage view with all variables as bands*

Configurations

From the configurations page, new configurations can be created from scratch or from templates (or copied from existing configurations), existing configurations can be edited and removed.

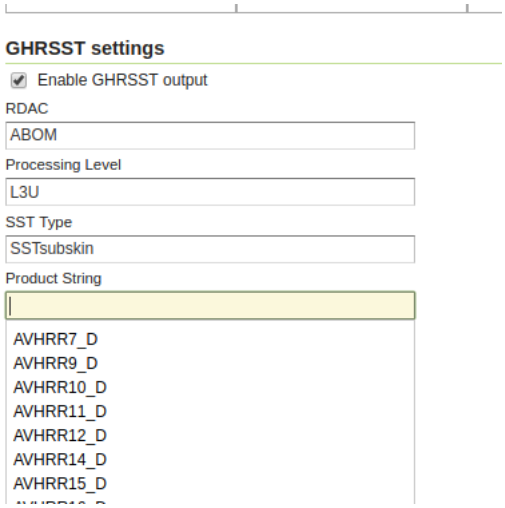


Fig. 16.24: *Enabling GHRSST output mode*

When removing a configuration, you have to option to do a *clean-up*, which will attempt to remove all resources (database tables, files, layers) that were created by (tasks of) this configuration. If this (partially) fails, the configuration will still be removed and the user will be notified.

Once you open a new or existing configuration, attributes, tasks and batches can be edited.

The attribute table adjusts automatically based on the information in the tasks table; and only the values must be filled in. In the task table, the name and parameters of each task can be edited, and new tasks can be created. Tasks can only be removed if they are not part of a batch any longer. Batches can only be removed if they are not running anywhere. When removing a task, you have to option to do a *clean-up*, which will attempt to remove all resources (database tables, files, layers) that were created by this task. If this (partially) fails, the task will still be removed and the user will be notified.

Batches can be created and edited from here as well, however the configuration must exist in order to be

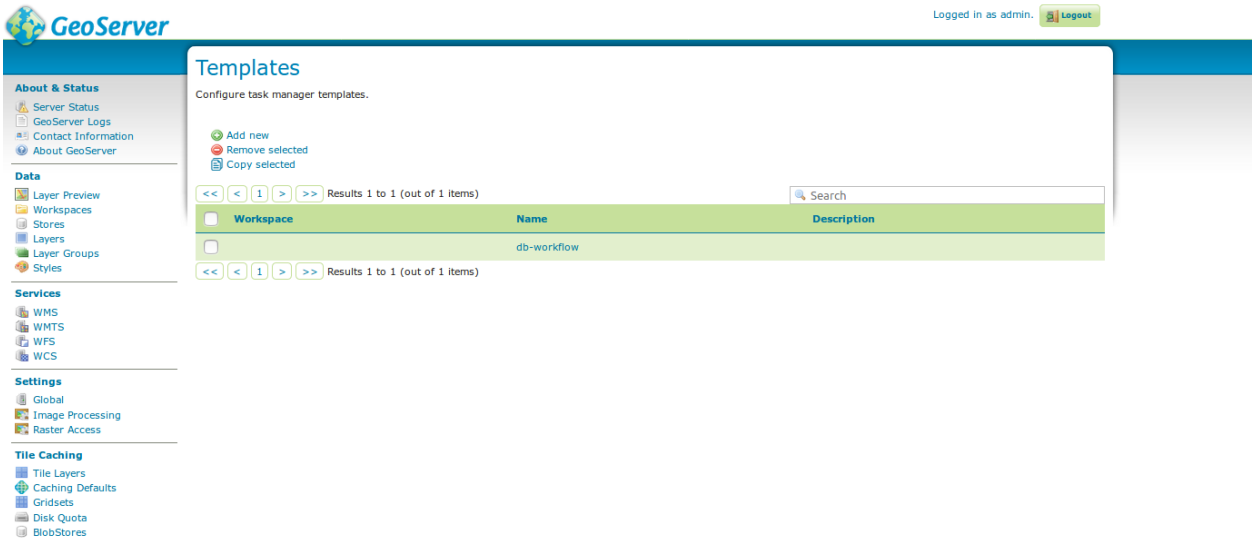


Fig. 16.25: templates

able to do that (in case of a new configuration, you must click `apply` once before you can create new batches). New tasks must also be saved (again, via the `apply` button) before they can be added to a batch. In case that the conditions are met, batch runs can be started, and the status/history of current and past batch runs can be displayed. Current batch runs can be interrupted (which is not guaranteed to happen immediately).

Batches

From the batches page, new independent batches (not associated with a configuration) can be created, existing batches can be edited and removed. All existing batches - independent as well as belonging to a configuration - are shown, unless they are special (if they start with a `@`) or if the configuration has not yet been completed (see initializing templates).

In case that the conditions are met, batch runs can be started, and the status/history of current and past batch runs can be displayed. Current batch runs can be interrupted (which is not guaranteed to happen immediately).

Once you open a new or existing batch, one can add or remove tasks from it and change the order of the tasks. You can also enable/disable the batch (if disabled, the batch is not scheduled) and choose the scheduling time. The user can choose between a daily schedule (with time), weekly (with day of week and time), monthly (with day of month and time) or specify a custom cron expression.

Task Types

- `CopyTableTask` Copy a database table from one database to another. The user can specify a source database, source table name, target database and target table name. Supports commit/rollback by creating a temporary table.
- `CreateViewTask` Create a view based on a single table. The user can specify the database, the table name, the selected fields and (optionally) a where condition. Supports commit/rollback by creating a temporary view.

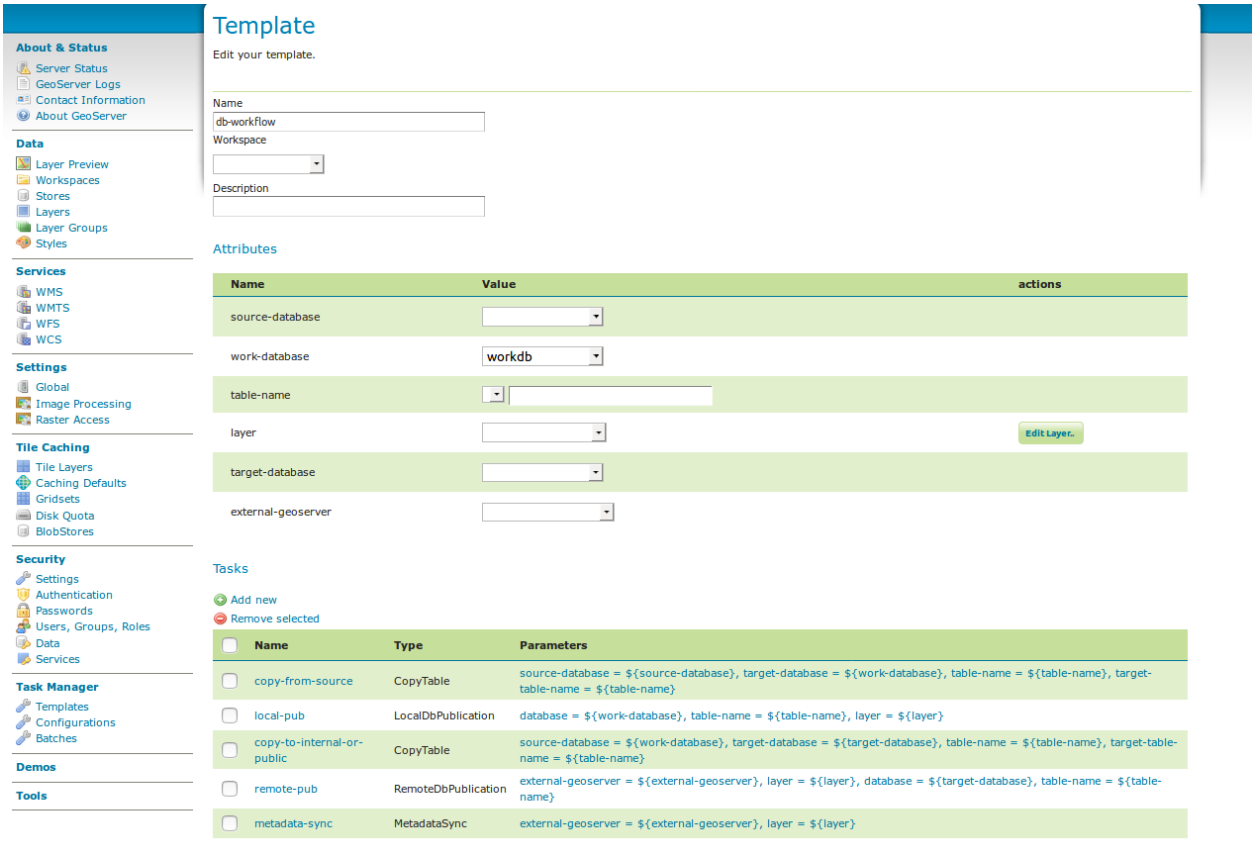


Fig. 16.26: template db workflow

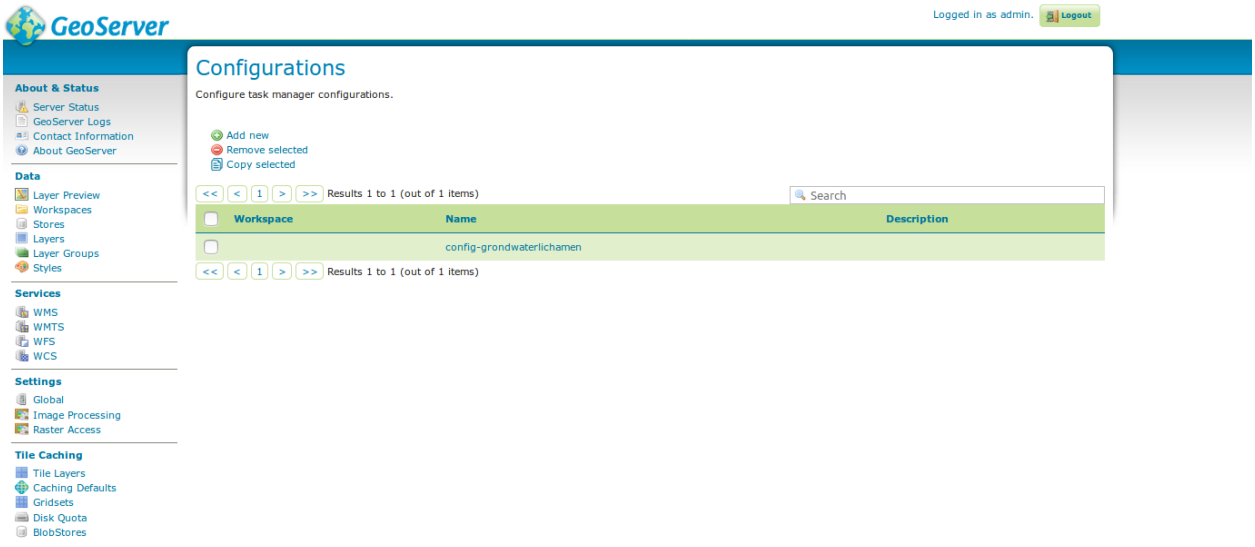


Fig. 16.27: configurations

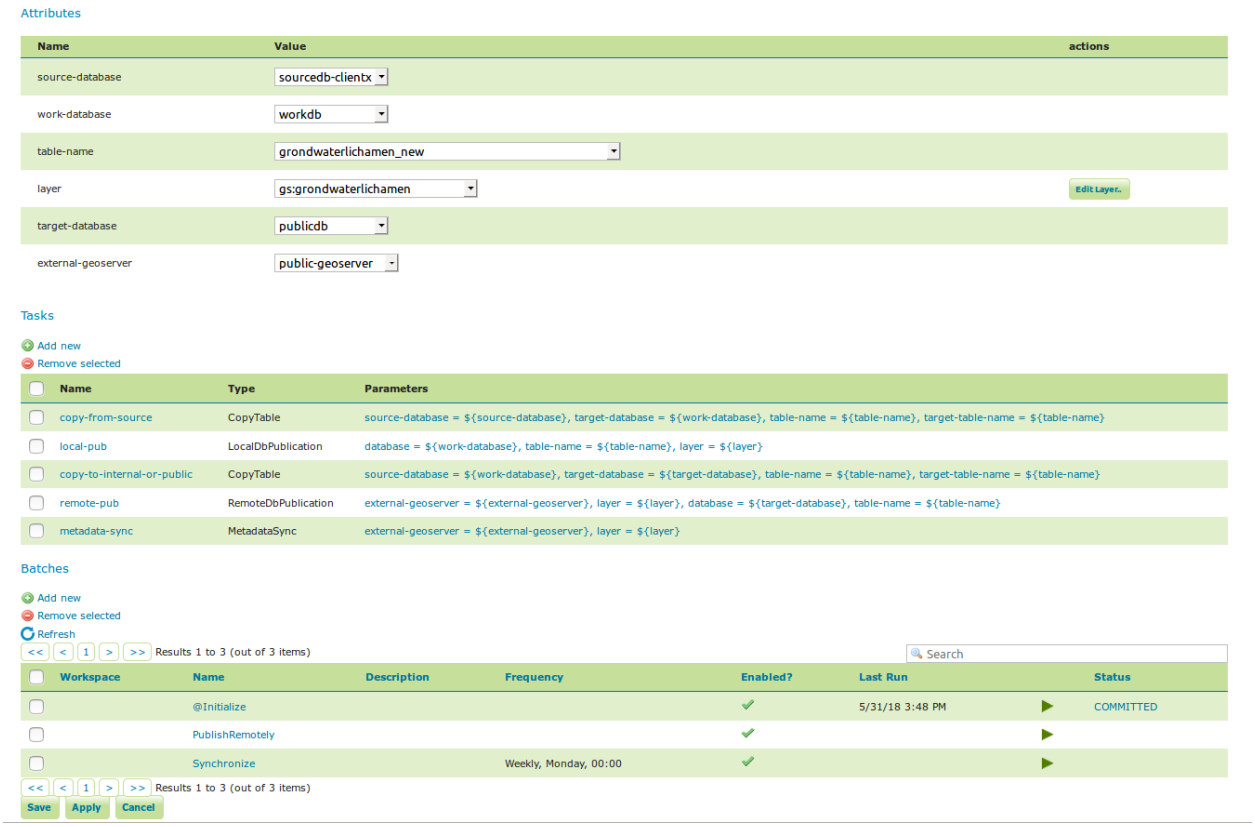


Fig. 16.28: workflow config 2

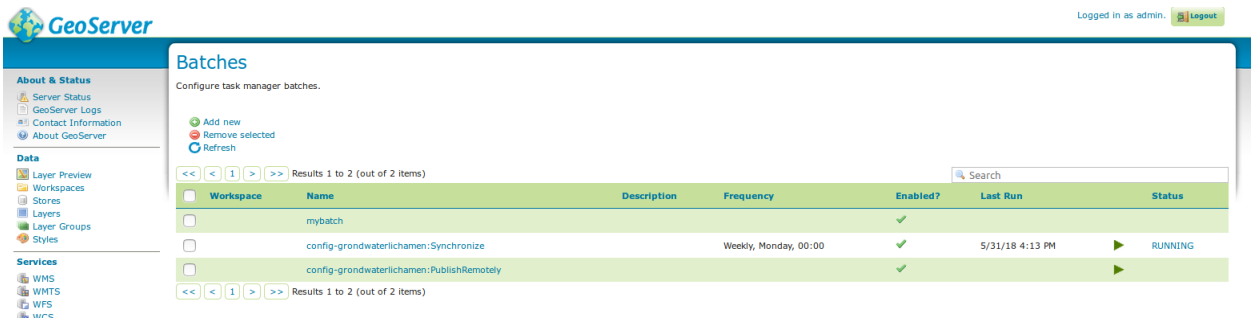


Fig. 16.29: batches

- `CreateComplexViewTask` Create a view based on a multiple tables. The user can specify the database and a whole query, where it can use any other configuration attribute in the form of '${placeholder}'. Supports commit/rollback by creating a temporary view.
- `CopyFileTask` Copy a file from one file service to another. Commit/rollback is supported by a versioning system, where the version of the file is inserted into the file name. The location of the version number is specified in the path as `###` (or set auto-versioned to `true` to add the placeholder automatically before the extension dot). On commit, the older version is removed. On rollback, the newer version is removed. The publication tasks will automatically publish the latest version.
- `LocalDbPublicationTask` Publish a database layer locally. The user can specify database, table and a layer name. Supports commit/rollback by advertising or removing the layer it created.
- `RemoteDbPublicationTask` Publish a database layer to another geoserver. The user can specify a target geoserver, a source layer and a target database. All information is taken from the source layer except for the target database which may be different. Supports commit/rollback through creating a temporary (unadvertised) layer. This task also supports the version place holder or auto-versioning, in order to combine with the `CopyFileTask`.
- `LocalFilePublicationTask` Publish a file layer locally (taster or shapefile). The user can specify a file service, a file (which can be uploaded unto the service) and a layer name. Supports commit/rollback by advertising or removing the layer it created.
- `RemoteFilePublicationTask` Publish a file layer locally (taster or shapefile). The user can specify a target geoserver, a source layer and a target file service and path (optional). All information is taken from the source layer except for the file service and path which may be different. Supports commit/rollback through creating a temporary (unadvertised) layer.
- `MetaDataSyncTask` Synchronise the metadata between a local layer and a layer on another geoserver (without re-publishing). The user can specify a target geoserver, a local and a remote layer. Does not support commit/rollback.
- `ConfigureCachedLayer` Configure caching for a layer on a remote geoserver with internal GWC, synchronise the settings with the local geoserver. This task may turn caching on or off depending on local configuration.
- `ClearCachedLayer` Clear (truncate) all tiles of a cached layer on a remote geoserver with internal GWC.

Import Tool

The import tool allows bulk creation of an unlimited amount of configurations on the basis of a template and a CSV file with attribute values. Contrary to the rest of the configuration, this function is only exposed via a REST service and not via the GUI. The import tool will generate a new configuration for each line in the CSV file, except for the first. The first line must specify the attribute names which should all match attributes that exist in the template, plus `name` (required), `description` (optional) and `workspace` (optional) for the configuration metadata. The CSV file mustspecify a valid attribute value for each required attribute.

To invoke the import tool, `POST` your CSV file to `http://{geoserver-host}/geoserver/taskmanager-import/{template}`

Optionally, you may specify the query parameter *validate=false* which will skip validation (at your own risk).

Examples

Consider the following setup.

Three geoservers:

- `work geoserver`: a geoserver only available in the local network, only used by administrators. New and updated data is published here as layers for the first time, to test both the validity of data and the publication configuration.
- `internal geoserver`: a geoserver only available in the local network, for internal users.
- `public geoserver`: a geoserver available on the internet, for the general public.

Several databases:

- `multiple source databases`: these are databases provided by partners that provide new and updated data. they are not used to directly publish on a geoserver.
- `work database`: database used by the `work geoserver` where its vector data is stored.
- `internal database`: database used by the `internal geoserver` where its vector data is stored.
- `public database`: database used by the `public geoserver` where its vector data is stored.

A typical workflow for a new layer goes as follows:

1. A new table is copied from a `source database` to the `work database` and then published on the `work geoserver`
2. After testing, the table is either copied to the `internal database` and published on the `internal geoserver` or copied to the `public database` and published on the `public geoserver`.
3. Every week, data is synchronised between the three databases and metadata is synchronised between the two geoservers.

Taskmanager should be installed only on the `work geoserver`. Then we could make the following template:

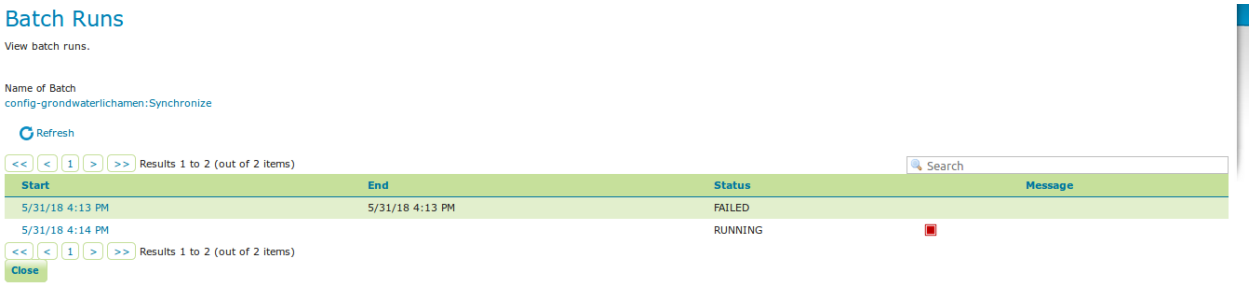


Fig. 16.30: batchruns

with the following batches:

The `@Initialize` batch:

The `PublishRemotely` batch:

The `Synchronize` batch:

When we now create a new configuration based on this template we choose a source database, table name and layer name:

After clicking apply, the configuration is being initialized (the layer is created locally)...

We can now fill in the rest of the details, save, and make the remote publication. The synchronization is scheduled weekly.

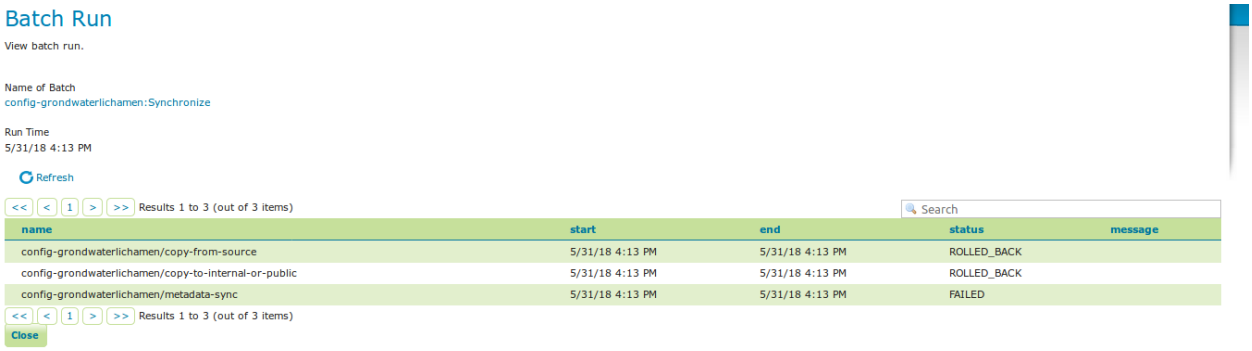


Fig. 16.31: batchrun

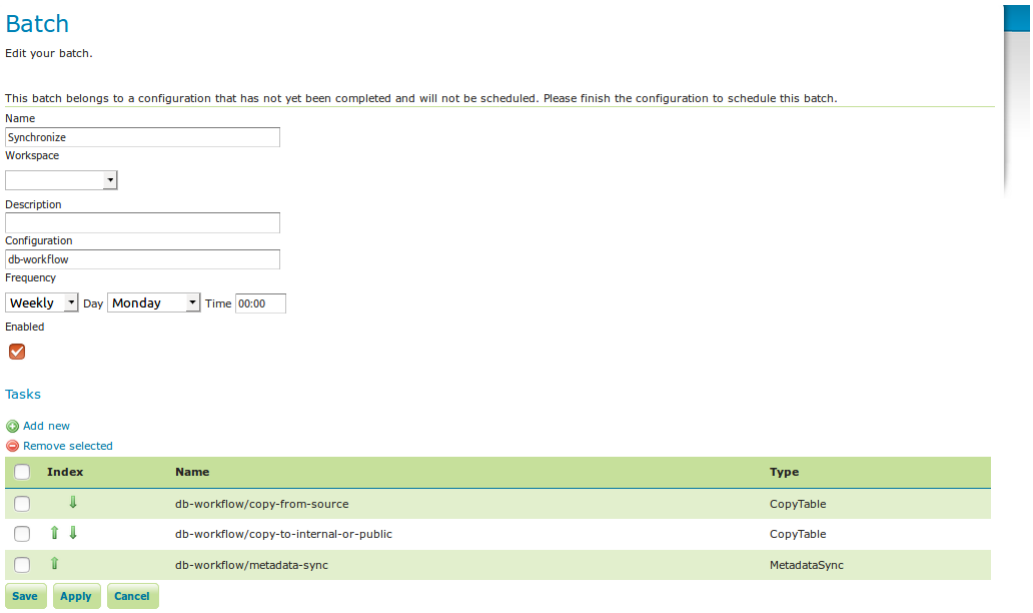


Fig. 16.32: batch synchronize

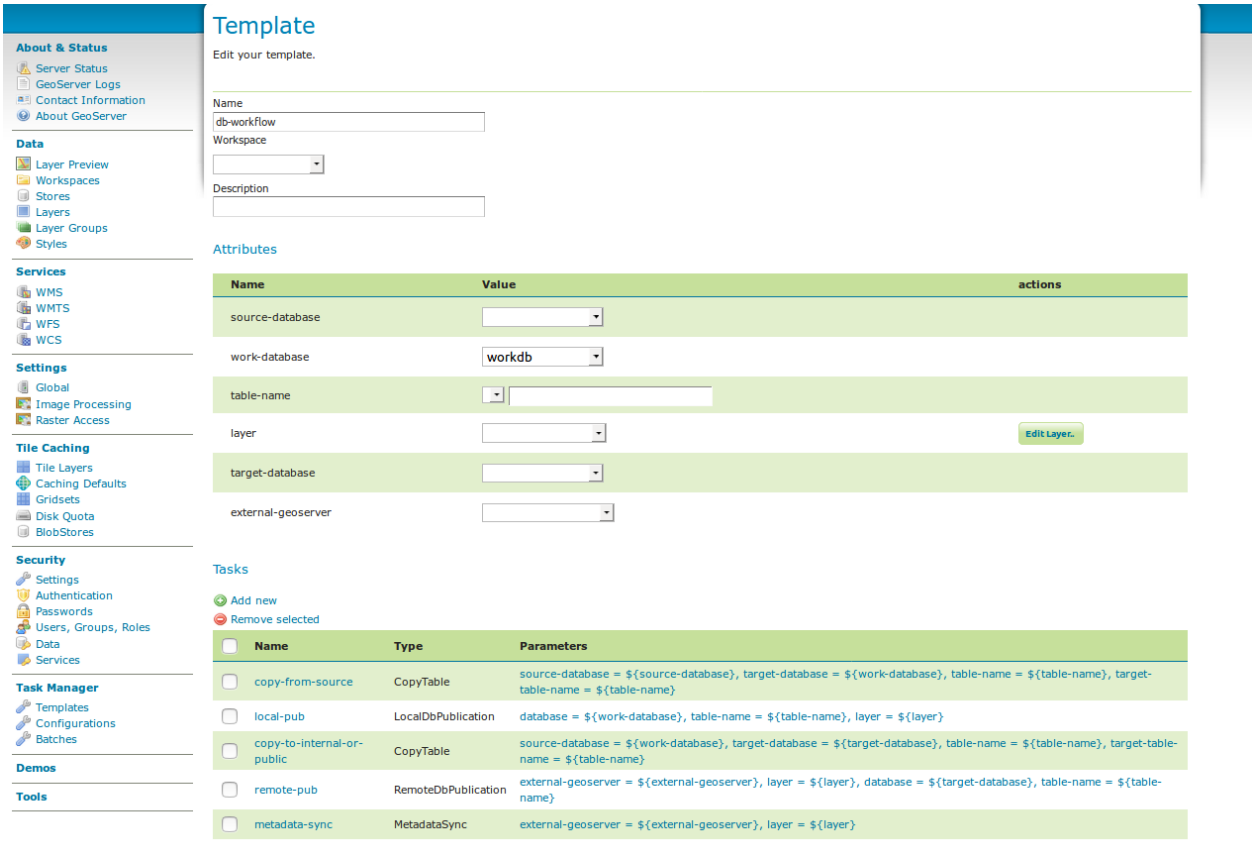


Fig. 16.33: template db workflow

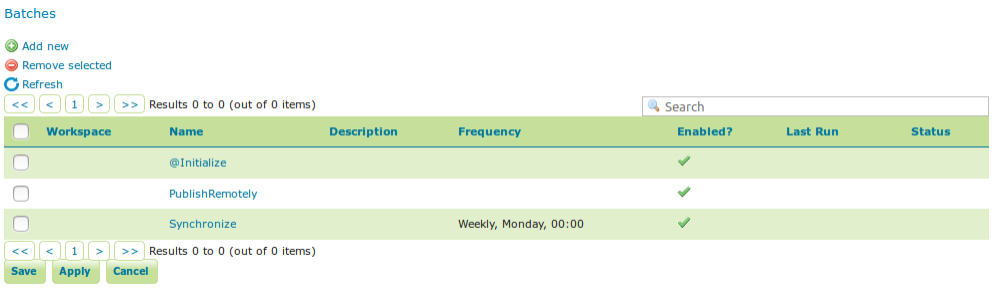


Fig. 16.34: template db workflow batches

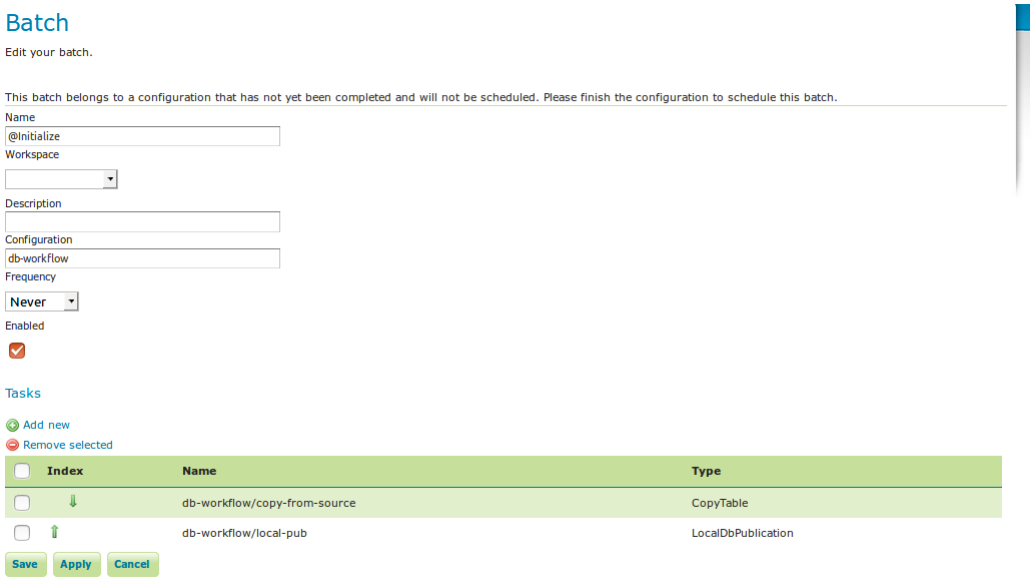


Fig. 16.35: batch initialize

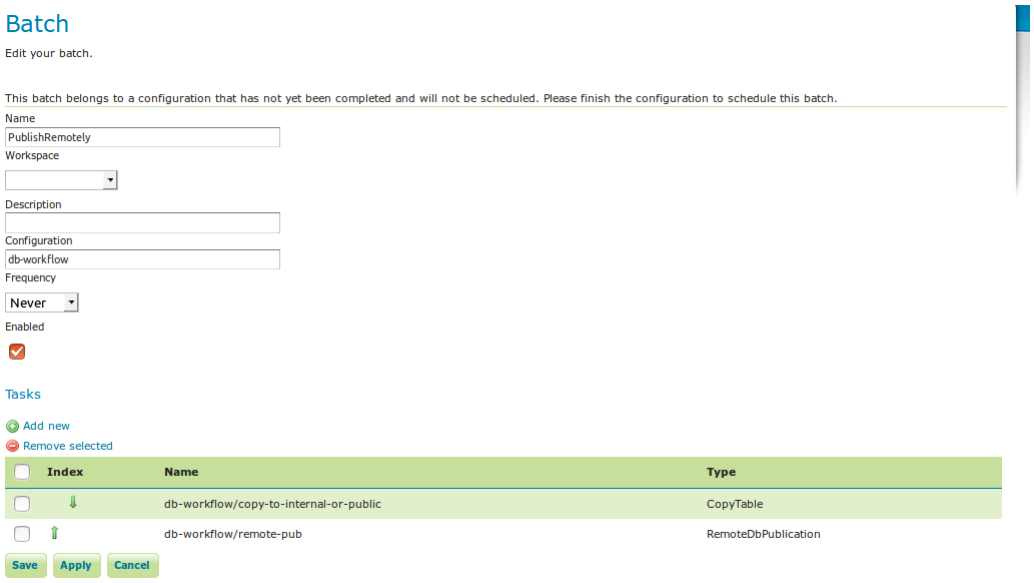


Fig. 16.36: batch publish remotely

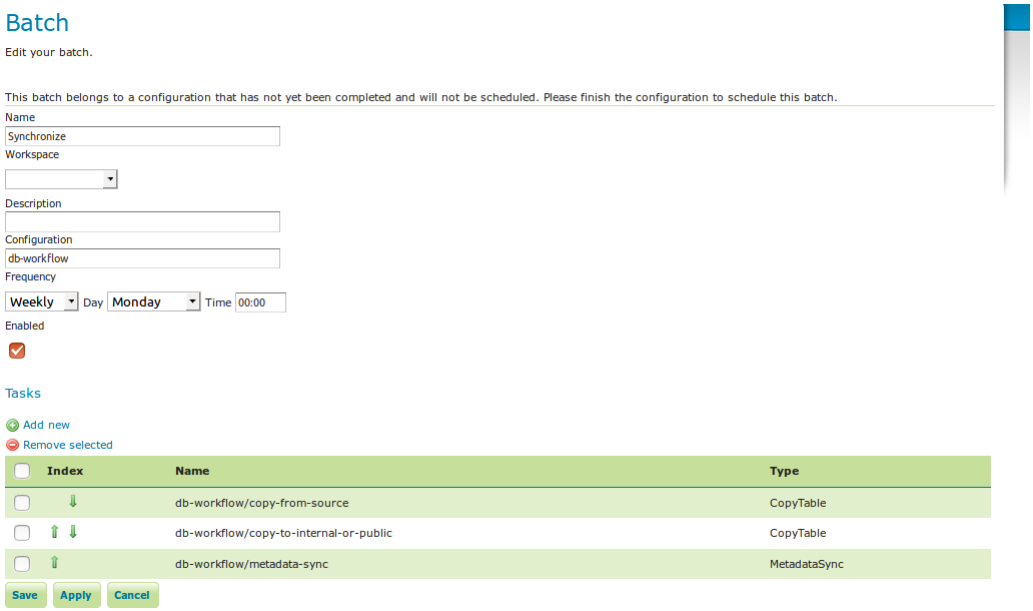


Fig. 16.37: batch synchronize

16.34.3 Developer's Guide

- *Task Types* - write your own operations
- *Actions* - extend the GUI for your tasks
- *Reporting* - choose the content and destination of your batch reports

Task Types

Task manager can be extended with custom made task types. Make your own implementation of the interface `TaskType` and let it be a spring bean. The name provided via the `Named` interface is used as a reference.

```
/**
 * A Task Type.
 *
 */
public interface TaskType extends Named {

    /**
     * Return parameter info for this task type.
     * It is recommended to use a LinkedHashMap and add the parameters in a intuitive␣
↪order.
     * This order will be preserved to present parameters to the user.
     *
     * @return the parameter info
     */
    Map<String, ParameterInfo> getParameterInfo();

    /**
     * Run a task, based on these parameter values.
     * @param ctx task context
```

```
     * @return the task result
     */
    TaskResult run(TaskContext ctx) throws TaskException;

    /**
     * Do a clean-up for this task (for example, if this task publishes something,␣
↪remove it).
     * @param ctx task context
     * @throws TaskException
     */
    void cleanup(TaskContext ctx) throws TaskException;

    /**
     * task type can specify whether it supports clean-up or not
     *
     * @return true if clean-up is supported
     */
    default boolean supportsCleanup() {
        return true;
    }

}
```

A ParameterInfo object contains a name, a *type*, whether they are required, and which other parameters they depend on (for example, a database table depends on a database).

The Task context looks as follows:

```
/**
 * Task Context used during batch run or task clean-up.
 *
 */
public interface TaskContext {

    /**
     * @return the task
     */
    Task getTask();

    /**
     * @return the batch context, null if this is a clean-up
     */
    BatchContext getBatchContext();

    /**
     *
     * @return the parameter values, lazy loaded from task and configuration.
     *
     * @throws TaskException
     */
    Map<String, Object> getParameterValues() throws TaskException;

    /**
     * Tasks can call this function to check if the user wants to interrupt the batch
     * and interrupt themselves.
     * If they do, they should still return a TaskResult that implements a roll back
     * of what was already done.
     *
```

```
     * @return whether the batch run should be interrupted, false if this is a clean-
↪up
     */
    boolean isInterruptMe();

}
```

The batch context looks as follows:

```
/**
 * During run, tasks create temporary objects that are committed to real objects␣
↪during
 * the commit phase (such as a table name) This maps real objects
 * to temporary objects during a single batch run. Tasks should save and look up␣
↪temporary
 * objects so that tasks within a batch can work together.
 *
 */
public interface BatchContext {

    public static interface Dependency {
        public void revert() throws TaskException;
    }

    Object get(Object original);

    Object get(Object original, Dependency dependency);

    /**
     * Whatever is put here in the task, must be removed in the commit!
     *
     * @param original
     * @param temp
     */
    void put(Object original, Object temp);

    void delete(Object original) throws TaskException;

    BatchRun getBatchRun();

}
```

The task result looks as follows:

```
/**
 * A handle of a task that was run but must still be committed or rolled back.
 *
 *
 */
public interface TaskResult {

    /**
     * finalize and clean-up resources any roll-back data
     */
    void commit() throws TaskException;

    /**
     * batch has failed - cancel all changes
```

```
     */
    void rollback() throws TaskException;

}
```

This is an example of how a task type can create temporary object:

```
//inside TaskType.run method

ctx.getBatchContext().put(originalObject, tempObject)

...

return new TaskResult() {
   @Override
   public void commit() throws TaskException {
       //this MUST be done!!!
       ctx.getBatchContext.delete(originalObject)
   }

       ...

}
```

Another task type would use this temporary object as follows:

```
 //inside TaskType.run method

Object tempObject = ctx.getBatchContext().get(originalObject, new Dependency() {
   @Override
   public void revert() {
       Object object = ctx.getBatchContext().get(originalObject);

       mySomething.setMyProperty(object);
       mySomething.save();
   }
});

mySomething.setMyProperty(tempObject);
mySomething.save();
```

Parameter Types

Custom task types may use existing or define new parameter types. They handle parameter validation, parsing parameter Strings into other object types, and provide information to the GUI about the parameters.

Existing regular Parameter Types (static members of `ParameterType` interface):

- `STRING`
- `INTEGER`
- `BOOLEAN`
- `URI`
- `SQL` (protects against ';' hacking)

External Parameter Types (members of `ExtTypes` spring bean): * `dbName`: database name * `tableName`: table name (parameter must depend on parameter of `dbName` type) * `extGeoserver`: external geoserver * `internalLayer`: layer from geoserver catalog * `name`: name qualified with namespace from geoserver catalog * `fileService`: file service * `file`: reference to file (parameter must dpend of parameter of `fileService` type)

Defining a new Parameter Type:

```
/**
 *
 * A Parameter Type For a Task
 *
 */
public interface ParameterType {

    /**
     * List possible values for this parameter (when applicable).
     * Include an empty string if custom value is also allowed.
     *
     * @param dependsOnRawValues raw values of depending parameters.
     * @return list of possible values, null if not applicable.
     */
    public List<String> getDomain(List<String> dependsOnRawValues);

    /**
     * Validate and parse a parameter value for this parameter (at run time).
     *
     * @param value the raw value.
     * @param dependsOnRawValues raw values of depending parameters.
     * @return the parsed value, NULL if the value is invalid.
     */
    public Object parse(String value, List<String> dependsOnRawValues);

    /**
     * Validate a parameter value (at configuration time).
     *
     * @param value the raw value.
     * @param dependsOnRawValues raw values of depending parameters.
     * @return true if the value is considered valid at configuration time (may still␣
↪be considered
     * invalid at parse time)
     */
    public default boolean validate(String value, List<String> dependsOnRawValues) {
        return parse(value, dependsOnRawValues) != null;
    }

    /**
     * Returns a list of web actions related to this type
     *
     * @return list of web actions
     */
    public default List<String> getActions() {
        return Collections.emptyList();
    }

}
```

Actions

Actions are extensions to the taskmanager webGUI attached to particular parameter types.

```
public interface Action extends Named, Serializable {

    /**
     * Execute this action.
     *
     * @param onPage the configuration page.
     * @param target the target of the ajax request that executed this action.
     * @param valueModel the value of the attribute, for reading and writing.
     * @param dependsOnRawValues raw values of depending attributes.
     */
    void execute(ConfigurationPage onPage, AjaxRequestTarget target, IModel<String>␣
↪valueModel, List<String> dependsOnRawValues);

    /**
     * Check whether this action can be executed with current values.
     * \
     * @param value the value of the attribute.
     * @param dependsOnRawValues raw values of depending attributes.
     * @return whether this action accepts these values.
     */
    default boolean accept(String value, List<String> dependsOnRawValues) {
        return true;
    }

}
```

In order to be linked to parameter types, an action must be spring bean. The name provided via the `Named` interface is used as a reference.

Reporting

Report builders

Reports are user friendly representations of finished batch runs, that are sent to some destination right after the batch run has finished. A report has a type (`FAILED`, `CANCELLED` or `SUCCESS`), a title and a content. Use spring to configure a single report builder.

```
/**
 * A report builder generates a report from a batch.
 * One could write a custom one.
 *
 */
public interface ReportBuilder {

    Report buildBatchRunReport(BatchRun batchRun);

}
```

Report services.

Use spring to configure any number of report services.

```
/**
 * A report service sends a report to a particular destination.
 * One can add an unlimited amount of report services which will all be used.
 *
 */
public interface ReportService {

    /**
     * Enumeration for filter.
     *
     */
    public enum Filter {
        /** All batch runs are reported **/
        ALL (Report.Type.FAILED, Report.Type.CANCELLED, Report.Type.SUCCESS),
        /** Only failed and cancelled batch runs are reported **/
        FAILED_AND_CANCELLED (Report.Type.FAILED, Report.Type.CANCELLED),
        /** Only failed batch runs are reported **/
        FAILED_ONLY (Report.Type.FAILED);

        Report.Type[] types;

        private Filter(Report.Type... types) {
            this.types = types;
        }

        public boolean matches(Report.Type type) {
            return ArrayUtils.contains(types, type);
        }

    }

    /**
     * Return the filter of the report.
     *
     * @return the filter of the report.
     */
    public Filter getFilter();

    /**
     * Send a report.
     *
     * @param report the report.
     */
    public void sendReport(Report report);

}
```

16.35 Quality of Service and Experience Module (QoSE)

The OGC standards for the `Quality of Service and Experience (QoSE)` module aim to improve the quality of the Web Services and APIs, including factors like availibility, capacity and performance by using well-defined metrics in order to provide comparable QoS measurements. The declared metadata in the Quality of Service and Experience contribute to get a complete picture of the Quality of Service of the entire Spatial Data Infrastructure, and more effectively maintain high Quality of Experience for its end users.

16.35.1 Installing the Plug-in

As a community module, the package needs to be downloaded from the nightly builds, picking the community folder of the corresponding GeoServer series (e.g. if working on GeoServer master nightly builds, pick the zip file form master/community-latest). The module is only supported in GeoServer 2.15 and later on.

To install the module, unpack the zip file contents into GeoServer own WEB-INF/lib directory and restart GeoServer.

16.35.2 Configuring QoSE

After the installation of the QoSE Plug-in, the WMS and WFS settings will show a configuration section for QoSE.

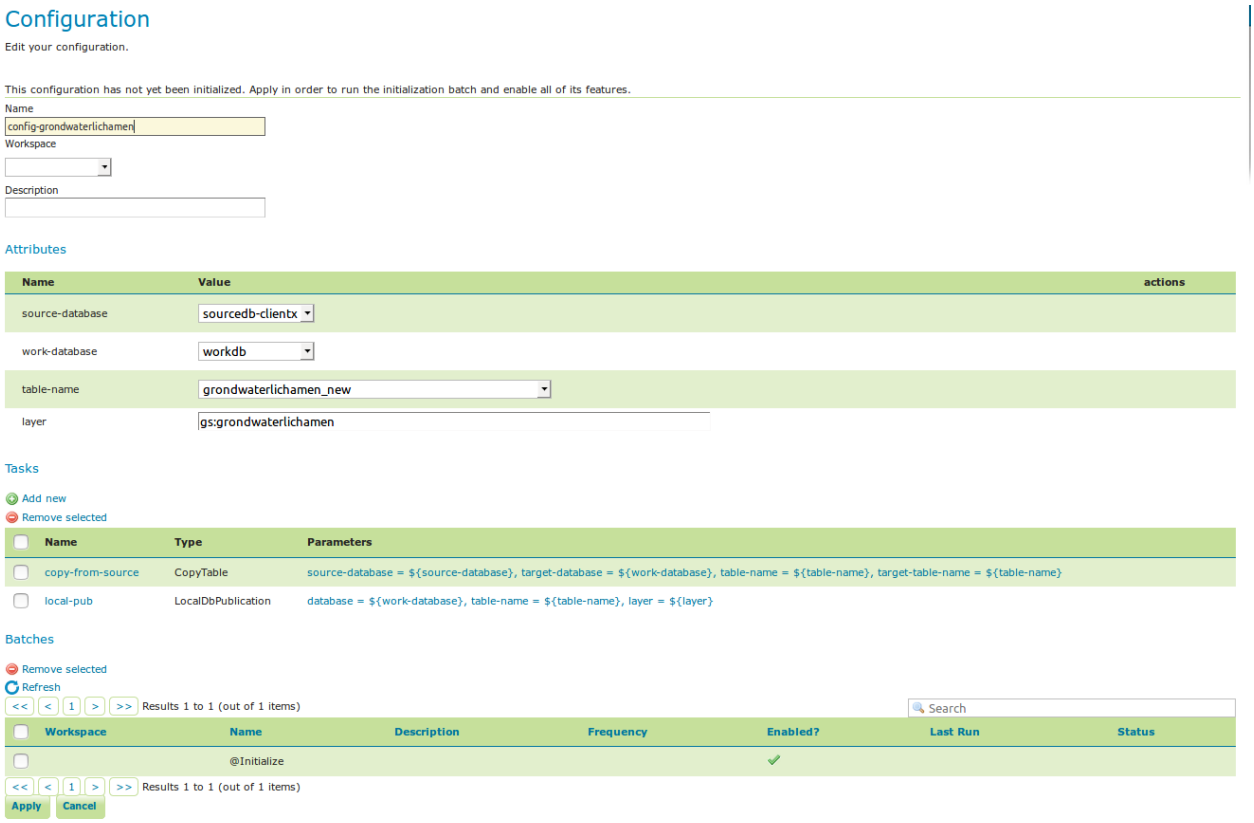


Fig. 16.38: workflow config

Once you enable the checkbox in the figure above, four additional sections will be shown.

Operating Info

The Operating Info section allow the user to define a schedule of the Operating Status of the Service. Many schedules with different Operating Status can be added. Specifically, the Operating Status menu list contains three options:

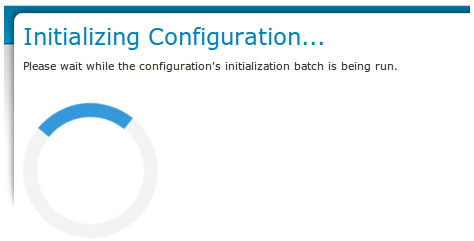


Fig. 16.39: initializing...

- *Operational:* The service or endpoint can be expected to function reliably and return correct and up-to-date information according to it's service and dataset descriptions.
- *PreOperational:* The service or endpoint may be under maintenance or down in the scheduled time.
- *NonOperational:* The service may be unavailable or unstable.

In the Operating Info Time, the user can set the days of the week and the Start/End time with the following format; hours:minutes:seconds+timezone (e.g. 01:00:00+03:00 or 09:00:00-04:00).

Note: Title, Start and End time are mandatory fields, and the user can add multiple time parts.

Statements

Through this section, the user can declare several performance statements of the service using various metrics such as the performance, the availibility and the capacity of the service.

Operation Anomaly Feed

The user can also provide the module with external resources or files that contain further info on the operation anomalies. For example, a calendar that indicates maintenance periods, downtimes or network slowness or a Log file.

Note: All the fields are mandatory. The URL field can accept filenames or relative/absolute URLs.

Representative Operations

In this section, the user can set the parameters of an operation such as a GetMap or GetFeatureInfo request to auto-configure external monitoring tools. Then, as a statement can declare the expected performance of the service for the latter monitored operation.

Once you have finished to set all the desired operations, you can click on submit.

The module will inject the settings in the XML GetCapabilities file of the service.

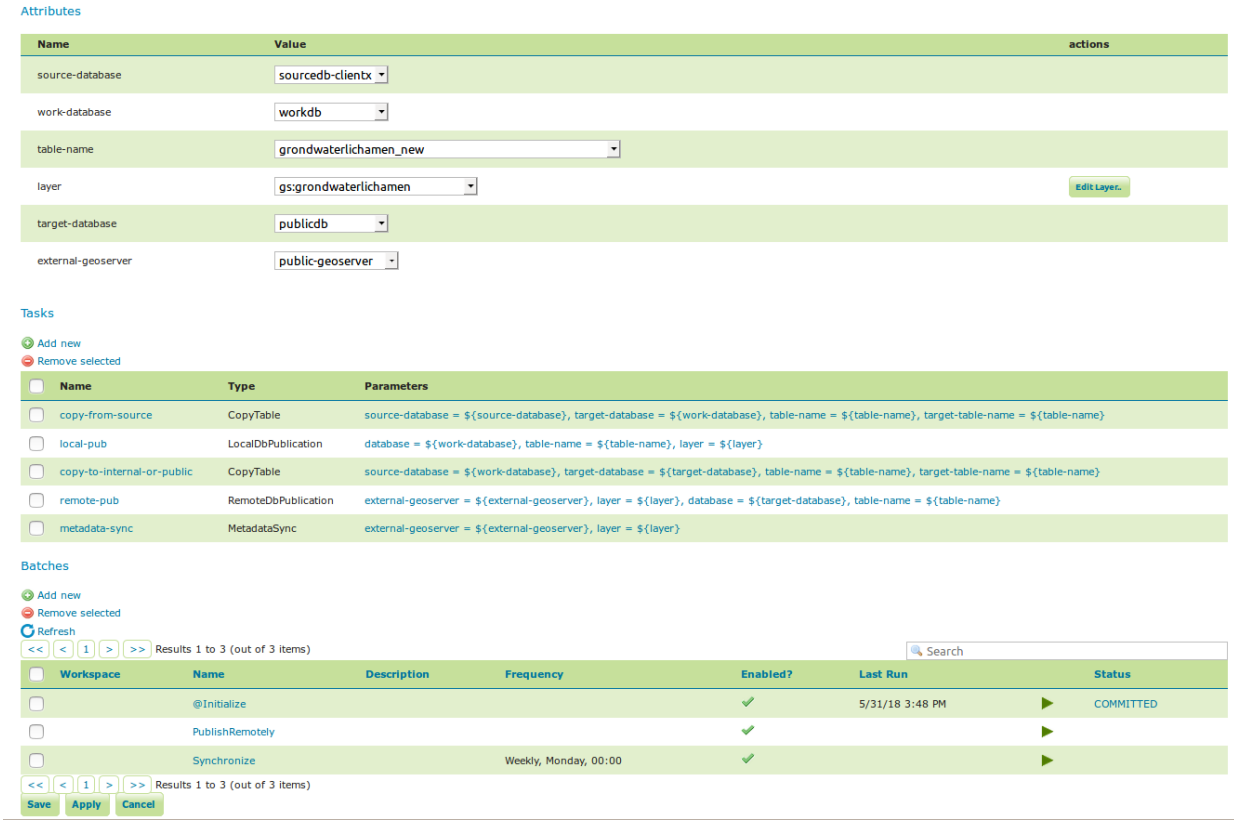
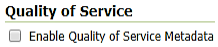
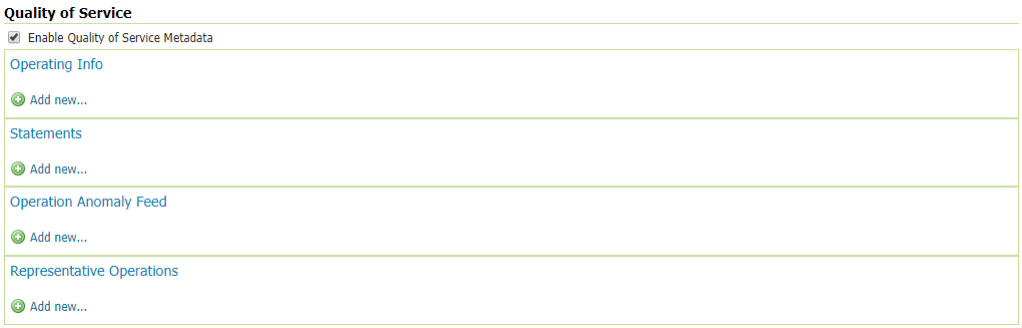
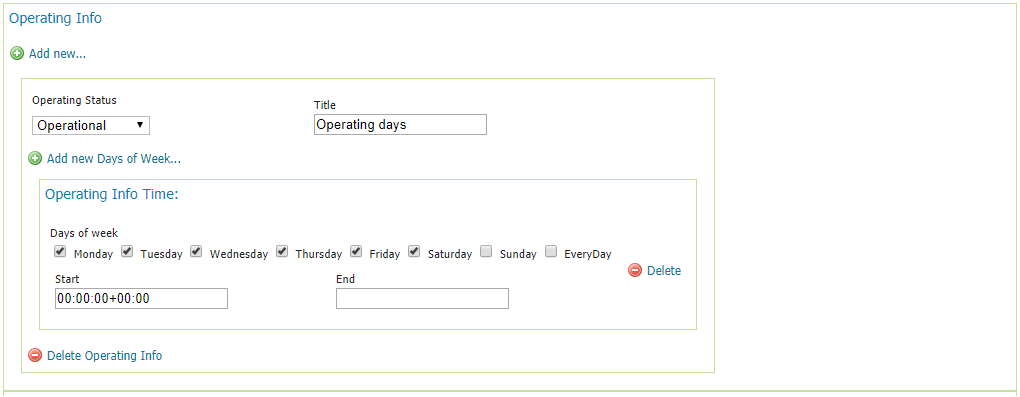


Fig. 16.40: workflow config 2








```
<qos:QualityOfServiceStatement>
     <qos:Metric xlink:href="http://def.opengeospatial.org/codelist/qos/metrics/1.0/
↪metrics.rdf#ResponseTime" xlink:title="Response Time"/>
     <qos:LessThanOrEqual uom="ms">500</qos:LessThanOrEqual>
   </qos:QualityOfServiceStatement>
   <qos:QualityOfServiceStatement>
     <qos:Metric xlink:href="http://def.opengeospatial.org/codelist/qos/metrics/1.0/
↪metrics.rdf#AvailabilityMonthly" xlink:title="Availability/Month"/>
     <qos:MoreThanOrEqual uom="%">95</qos:MoreThanOrEqual>
   </qos:QualityOfServiceStatement>
   <qos:RepresentativeOperation>
     <qos-wms:GetMapOperation>
       <ows:DCP>
         <ows:HTTP>
           <ows:Get/>
         </ows:HTTP>
       </ows:DCP>
       <qos-wms:RequestOption>
         <qos:AreaConstraint srsName="EPSG:4326">
           <qos:LowerCorner>-124.73142200000001 24.955967</qos:LowerCorner>
           <qos:UpperCorner>-66.969849 49.371735</qos:UpperCorner>
         </qos:AreaConstraint>
         <qos:RequestParameterConstraint name="LayerName">
           <ows:AllowedValues>
             <ows:Value>topp:states</ows:Value>
           </ows:AllowedValues>
         </qos:RequestParameterConstraint>
         <qos:RequestParameterConstraint name="CRS">
           <ows:AllowedValues>
             <ows:Value>EPSG:4326</ows:Value>
           </ows:AllowedValues>
         </qos:RequestParameterConstraint>
         <qos:RequestParameterConstraint name="OutputFormat">
           <ows:AllowedValues>
             <ows:Value>image/png</ows:Value>
           </ows:AllowedValues>
         </qos:RequestParameterConstraint>
         <qos:RequestParameterConstraint name="ImageWidth">
           <ows:AllowedValues>
             <ows:Range>
               <ows:MinimumValue>256</ows:MinimumValue>
               <ows:MaximumValue>500</ows:MaximumValue>
             </ows:Range>
```

```
          </ows:AllowedValues>
        </qos:RequestParameterConstraint>
        <qos:RequestParameterConstraint name="ImageHeight">
          <ows:AllowedValues>
            <ows:Range>
              <ows:MinimumValue>256</ows:MinimumValue>
              <ows:MaximumValue>500</ows:MaximumValue>
            </ows:Range>
          </ows:AllowedValues>
        </qos:RequestParameterConstraint>
      </qos-wms:RequestOption>
    </qos-wms:GetMapOperation>
    <qos:QualityOfServiceStatement>
      <qos:Metric xlink:href="http://def.opengeospatial.org/codelist/qos/metrics/1.0/
↪metrics.rdf#RequestResponsePerformance" xlink:title="GetMap Responce Performance␣
↪for layer States"/>
      <qos:LessThanOrEqual uom="s">2</qos:LessThanOrEqual>
    </qos:QualityOfServiceStatement>
  </qos:RepresentativeOperation>
  <qos:OperationAnomalyFeed xlink:href="http://monitoring.geo-solutions.it/resource/
↪65?lang=en">
    <ows:Abstract>Live Monitoring of the Service</ows:Abstract>
    <ows:Format>html</ows:Format>
  </qos:OperationAnomalyFeed>
</qos-wms:QualityOfServiceMetadata>
```

Note: The module works only for the latest versions of WMS (i.e. 1.3.0) and WFS (i.e. 2.0.2 later on) services.

CHAPTER 17

Tutorials

17.1 Freemarker Templates

17.1.1 Introduction

This tutorial will introduce you to a more in depth view of what FreeMarker templates are and how you can use the data provided to templates by GeoServer.

Freemarker is a simple yet powerful template engine that GeoServer uses whenever developer allowed user customization of outputs. In particular, at the time of writing it's used to allow customization of GetFeatureInfo, GeoRSS and KML outputs.

Freemarker allows for simple variable expansions, as in `${myVarName}`, expansion of nested properties, such as in `${feature.myAtt.value}`, up to little programs using loops, ifs and variables. Most of the relevant information about how to approach template writing is included in the Freemarker's Designer guide and won't be repeated here: the guide, along with the *KML Placemark Templates* and *GetFeatureInfo Templates* tutorials should be good enough to give you a good grip on how a template is built.

Template Lookup

GeoServer looks up templates in three different places, allowing for various level of customization. For example given the `content.ftl` template used to generate WMS GetFeatureInfo content:

- Look into `GEOSERVER_DATA_DIR/workspaces/<workspace>/<datastore>/<featuretype>/content.ftl` to see if there is a feature type specific template
- Look into `GEOSERVER_DATA_DIR/workspaces/<workspace>/<datastore>/content.ftl` to see if there is a store specific template
- Look into `GEOSERVER_DATA_DIR/workspaces/<workspace>/content.ftl` to see if there is a workspace specific template
- Look into `GEOSERVER_DATA_DIR/workspaces/content.ftl` looking for a global override
- Look into `GEOSERVER_DATA_DIR/templates/content.ftl` looking for a global override

- Look into the GeoServer classpath and load the default template

Each templated output format tutorial should provide you with the template names, and state whether the templates can be type specific, or not. Missing the source for the default template, look up for the service jar in the geoserver distribution (for example, wms-x.y.z.jar), unpack it, and you'll find the actual xxx.ftl files GeoServer is using as the default templates.

Common Data Models

Freemarker calls "data model" the set of data provided to the template. Each output format used by GeoServer will inject a different data model according to the informations it's managing, yet there are three very common elements that appear in almost each template, Feature, FeatureType and FeatureCollection. Here we provide a data model of each.

The data model is a sort of a tree, where each element has a name and a type. Besides basic types, we'll use:

- list: a flat list of items that you can scan thru using the FreeMarker `<#list>` directive;
- map: a key/value map, that you usually access using the dot notation, as in `${myMap.myKey}`, and can be nested;
- listMap: a special construct that is, at the same time, a Map, and a list of the values.

Here are the data models (as you can see there are redundancies, in particular in attributes, we chose this approach to make template building easier):

FeatureType (map)

- name (string): the type name
- attributes (listMap): the type attributes
  - name (string): attribute name
  - namespace (string): attribute namespace URI
  - prefix (string): attribute namespace prefix
  - type (string): attribute type, the fully qualified Java class name
  - isGeometry (boolean): true if the attribute is geometric, false otherwise

Feature (map)

- fid (string): the feature ID (WFS feature id)
- typeName (string): the type name
- attributes (listMap): the list of attributes (both data and metadata)
  - name (string): attribute name
  - namespace (string): attribute namespace URI
  - prefix (string): attribute namespace prefix
  - isGeometry (boolean): true if the attribute is geometric, false otherwise
  - value: a string representation of the the attribute value
  - isComplex (boolean): true if the attribute is a feature (see *Complex Features*), false otherwise
  - type (string or FeatureType): attribute type: if isComplex is false, the fully qualified Java class name; if isComplex is true, a FeatureType
  - rawValue: the actual attribute value (is isComplex is true rawValue is a Feature)

- type (map)
  - name (string): the type name (same as typeName)
  - namespace (string): attribute namespace URI
  - prefix (string): attribute namespace prefix
  - title (string): The title configured in the admin console
  - abstract (string): The abstract for the type
  - description (string): The description for the type
  - keywords (list): The keywords for the type
  - metadataLinks (list): The metadata URLs for the type
  - SRS (string): The layer's SRS
  - nativeCRS (string): The layer's coordinate reference system as WKT

FeatureCollection (map)

- features (list of Feature, see above)
- type (FeatureType, see above)

request (map)

Contains the GetFeatureInfo request parameters and related values.

environment (map)

Allows accessing several environment variables, in particular those defined in:

- JVM system properties
- OS environment variables
- web.xml context parameters

Examples

request

- ${request.LAYERS}
- ${request.ENV.PROPERTY}

environment

- ${environment.GEOSERVER_DATA_DIR}
- ${environment.WEB_SITE_URL}

17.2 GeoRSS

GeoServer supports GeoRSS as an output format allowing you to serve features as an RSS feed.

17.2.1 Quick Start

If you are using a web browser which can render rss feeds simply visit the url http://localhost:8080/geoserver/wms/reflect?layers=states&format=rss in your browser. This is assuming a local GeoServer instance is running with an out of the box configuration. You should see a result that looks more or less like this:

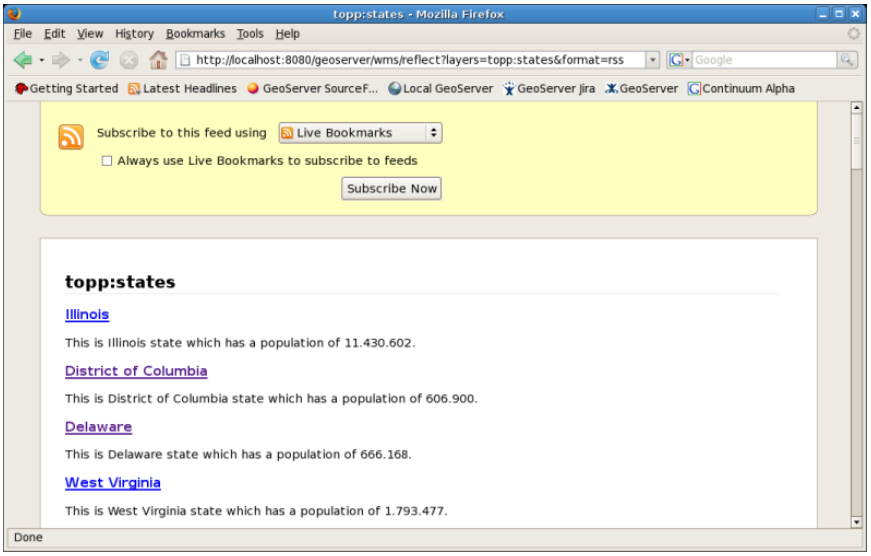


Fig. 17.1: *topp:states rss feed*

17.2.2 Templating

GeoServer uses freemarker templates to customize the returned GeoRSS feed. If you are not familiar with freemarker templates you may wish to read the *Freemarker Templates* tutorial, and the *KML Placemark Templates* page, which has simple examples.

Three template files are currently supported:

- `title.ftl`
- `description.ftl`
- `link.ftl`

Each of these files may be used to customize the assocated field in the GeoRSS feed.

17.2.3 Ajax Map Mashups

Note: For Ajax map mashups to work, the GeoServer instance must be visible to the Internet (i.e. using the address `localhost` will not work).

17.2.4 Google Maps

How to create a Google Maps mashup with a GeoRSS overlay produced by GeoServer.

1. Obtain a Google Maps API Key from Google.
2. Create an html file called `gmaps.html`:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org    R/
↪xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
    <meta http-equiv="content-type" content="text/html; charset=utf-8"/>
    <title>Google Maps JavaScript API Example<    itle>
    <script src="http://maps.google.com/maps?file=api&amp;v=2.x&amp;key=<INSERT␣
↪MAPS API KEY HERE>" type="text/javascript"></script>

    <script type="text/javascript">
    //<![CDATA[
        function load() {
            if (GBrowserIsCompatible()) {
                var map = new GMap2(document.getElementById("map"));
                map.addControl(new GLargeMapControl());
                map.setCenter(new GLatLng(40,-98), 4);
                var geoXml = new GGeoXml("<INSERT GEOSERVER URL HERE>/geoserver/
↪wms/reflect?layers=states&format=rss");
                map.addOverlay(geoXml);
            }
        }
    //]]>
    </script>

    </head>
    <body onload="load()" onunload="GUnload()">
        <div id="map" style="width: 800px; height: 600px"></div>
    </body>
</html>
```

3. Visit `gmaps.html` in your web browser.

Note: The version of the google maps api must be **2.x**, and not just **2** You must insert your specific maps api key, and geoserver base url

17.2.5 Yahoo Maps

How to create a Yahoo! Maps mashup with a GeoRSS overlay produced by GeoServer.

1. Obtain a <Yahoo Maps Application ID <http://search.yahooapis.com/webservices/register_application>'_ from Yahoo.
2. Create an html file called `ymaps.html`:

```
<html>
    <head>
    <title>Yahoo! Maps GeoRSS Overlay Example<    itle>
    <script src="http://api.maps.yahoo.com/ajaxymap?v=3.0&appid=<INSERT␣
↪APPLICATION ID HERE>" type="text/javascript"></script>
    <script type="text/javascript" language="JavaScript">

        function StartYMap() {
```

```
            var map = new YMap(document.getElementById('ymap'));
            map.addPanControl();
            map.addZoomShort();

            function doStart(eventObj) {
                var defaultEventObject = eventObj;
                //eventObj.ThisMap [map object]
                //eventObj.URL [argument]
                //eventObj.Data [processed input]
            }

            function doEnd(eventObj) {
                var defaultEventObject = eventObj;
                //eventObj.ThisMap [map object]
                //eventObj.URL [argument]
                //eventObj.Data [processed input]
                map.smoothMoveByXY(new YCoordPoint(10,50));
            }

            YEvent.Capture(map,EventsList.onStartGeoRSS, function(eventObj) {␣
↪doStart(eventObj); });
            YEvent.Capture(map,EventsList.onEndGeoRSS, function(eventObj) {␣
↪doEnd(eventObj); });

            map.addOverlay(new YGeoRSS('http://<INSERT GEOSERVER URL HERE>/
↪geoserver/wms/reflect?layers=states&format=rss'));
        }

    window.onload = StartYMap;
     </script>
      </head>
      <body>
           <div id="ymap" style="width: 800px; height: 600px; left:2px; top:2px">
↪</div>
     </body>
</html>
```

3. Visit `ymaps.html` in your web browser.

Note: The version of the yahoo maps api must be **3.0** You must insert your specific application id, and geoserver base url

17.2.6 Microsoft Virtual Earth

Note: Non Internet Explorer Users*: GeoRSS overlays are only supported in Internet Explorer, versions greater then 5.5.

How to create a Microsoft Virtual Earth mashup with a GeoRSS overlay produced by GeoServer.

Note: To access a GeoRSS feed from Microsoft Virtual Earth the file (ve.html) must be accessed from a Web Server, IE. It will not work if run from local disk.

1. Create an html file called `ve.html`. **Note**: You must insert your specific maps api key, and geoserver base url:

```
<html>
  <head>
    <script src="http://dev.virtualearth.net/mapcontrol/v4/mapcontrol.js"></
↪script>
    <script>
     var map;

     function OnPageLoad()
     {
        map = new VEMap('map');
        map.LoadMap();

        var veLayerSpec = new VELayerSpecification();
        veLayerSpec.Type = VELayerType.GeoRSS;
        veLayerSpec.ID = 'Hazards';
    veLayerSpec.LayerSource = 'http://<INSERT GEOSERVER URL HERE>/geoserver/wms/
↪reflect?layers=states&format=rss';
    veLayerSpec.Method = 'get';
    map.AddLayer(veLayerSpec);
    }
   </script>
 </head>
 <body onload="OnPageLoad();">
    <div id="map" style="position:relative;width:800px;height:600px;"></div>
  </body>

</html>
```

2. Visit `ve.html` in your web browser. You should see the following:

17.3 GetFeatureInfo Templates

This tutorial describes how to use the GeoServer template system to create custom HTML GetFeatureInfo responses.

17.3.1 Introduction

GetFeatureInfo is a WMS standard call that allows one to retrieve information about features and coverages displayed in a map. The map can be composed of various layers, and GetFeatureInfo can be instructed to return multiple feature descriptions, which may be of different types. GetFeatureInfo can generate output in various formats: GML2, plain text and HTML. Templating is concerned with the HTML one.

The default HTML output is a sequence of titled tables, each one for a different layer. The following example shows the default output for the tiger-ny basemap (included in the above cited releases, and onwards).

17.3.2 Standard Templates

The following assumes you're already up to speed with Freemarker templates. If you're not, read the *Freemarker Templates* tutorial, and the *KML Placemark Templates* page, which has simple examples.

The default output is generated by the standard templates, which are three:

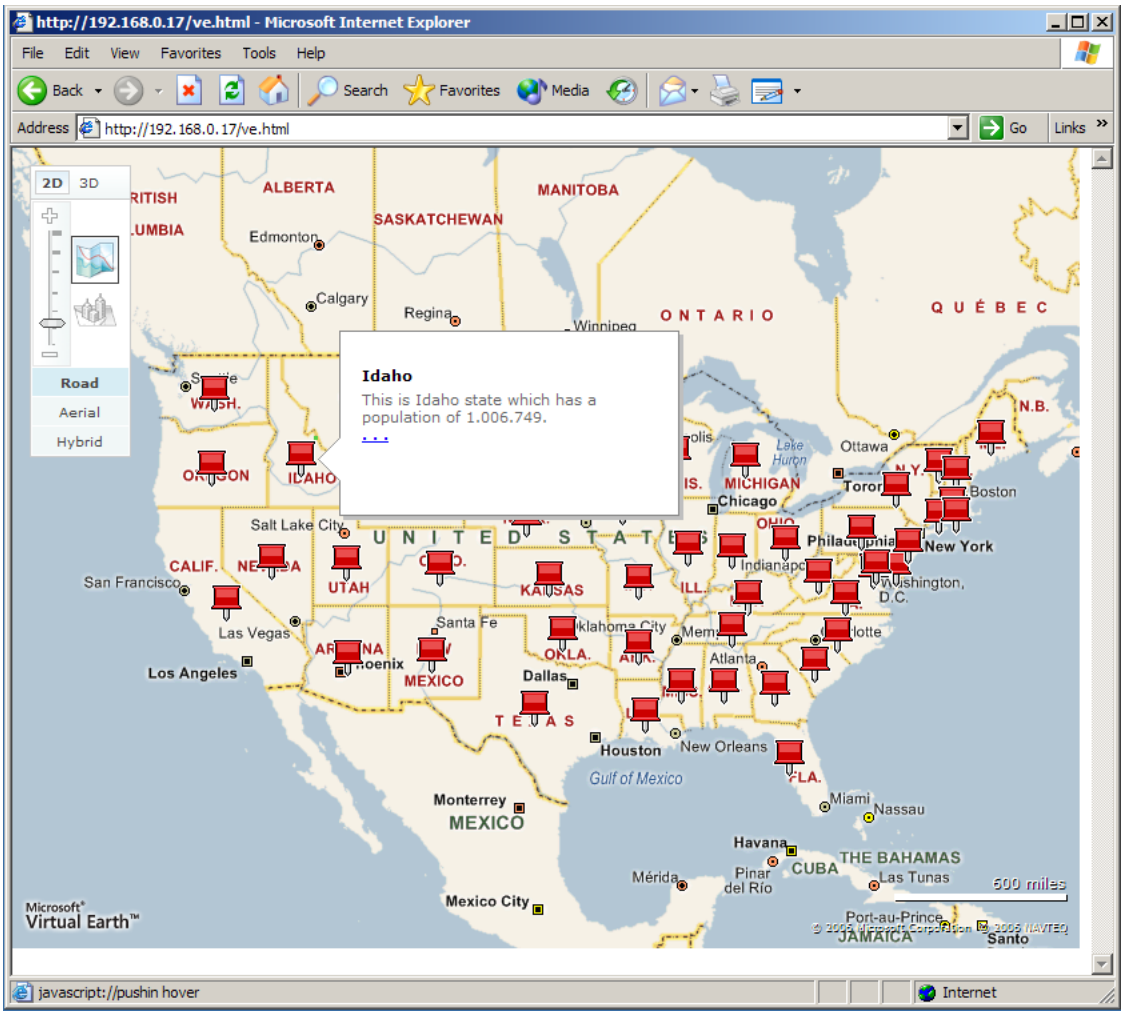


Fig. 17.2: *Virtual Earth*

Fig. 17.3: *Default Output*

- `header.ftl`
- `content.ftl`
- `footer.ftl`

The *header template* is invoked just once, and usually contains the start of the HTML page, along with some CSS. The default header template looks like this (as you can see, it's completely static, and it's in fact not provided with any variable you could expand):

```
<#--
Header section of the GetFeatureInfo HTML output. Should have the <head> section, and
a starter of the <body>. It is advised that eventual css uses a special class for␣
↪featureInfo,
since the generated HTML may blend with another page changing its aspect when usign␣
↪generic classes
like td, tr, and so on.
-->
<html>
  <head>
    <title>GeoServer GetFeatureInfo output</title>
  </head>
  <style type="text/css">
        table.featureInfo, table.featureInfo td, table.featureInfo th {
                border:1px solid #ddd;
                border-collapse:collapse;
                margin:0;
                padding:0;
                font-size: 90%;
                padding:.2em .1em;
        }
```

```
        table.featureInfo th{
            padding:.2em .2em;
                text-transform:uppercase;
                font-weight:bold;
                background:#eee;
        }
        table.featureInfo td{
                background:#fff;
        }
        table.featureInfo tr.odd td{
                background:#eee;
        }
        table.featureInfo caption{
                text-align:left;
                font-size:100%;
                font-weight:bold;
                text-transform:uppercase;
                padding:.2em .2em;
        }
  </style>
  <body>
```

The *footer template* is similar, a static template used to close the HTML document properly:

```
<#--
Footer section of the GetFeatureInfo HTML output. Should close the body and the html␣
↪tag.
-->
  </body>
</html>
```

The *content template* is the one that turns feature objects into actual HTML tables. The template is called multiple times: each time it's fed with a different feature collection, whose features all have the same type. In the above example, the template has been called once for the roads, and once for the points of interest (POI). Here is the template source:

```
<#--
Body section of the GetFeatureInfo template, it's provided with one feature␣
↪collection, and
will be called multiple times if there are various feature collections
-->
<table class="featureInfo">
  <caption class="featureInfo">${type.name}</caption>
  <tr>
<#list type.attributes as attribute>
  <#if !attribute.isGeometry>
    <th >${attribute.name}</th>
  </#if>
</#list>
  </tr>

<#assign odd = false>
<#list features as feature>
  <#if odd>
    <tr class="odd">
  <#else>
    <tr>
  </#if>
```

```
  <#assign odd = !odd>

  <#list feature.attributes as attribute>
    <#if !attribute.isGeometry>
      <td>${attribute.value}</td>
    </#if>
  </#list>
  </tr>
</#list>
</table>
<br/>
```

As you can see there is a first loop scanning type and outputting its attributes into the table header, then a second loop going over each feature in the collection (features). From each feature, the attribute collections are accessed to dump the attribute value. In both cases, geometries are skipped, since there is not much point in including them in the tabular report. In the table building code you can also see how odd rows are given the "odd" class, so that their background colors improve readability.

17.3.3 Custom Templates

So, what do you have to do if you want to override the custom templates? Well, it depends on which template you want to override.

`header.ftl` and `footer.ftl` are type independent, so if you want to override them you have to place a file named `header.ftl` or `footer.ftl` in the `templates` directory, located in your GeoServer *GeoServer data directory*. On the contrary, `content.ftl` may be generic, or specific to a feature type.

For example, let's say you would prefer a bulleted list appearance for your feature info output, and you want this to be applied to all GetFeatureInfo HTML output. In that case you would drop the following `content.ftl` in the templates directory:

```
<ul>
<#list features as feature>
  <li><b>Type: ${type.name}</b> (id: <em>${feature.fid}</em>):
  <ul>
  <#list feature.attributes as attribute>
    <#if !attribute.isGeometry>
      <li>${attribute.name}: ${attribute.value}</li>
    </#if>
  </#list>
  </ul>
  </li>
</#list>
</ul>
```

With this template in place, the output would be:

Looking at the output we notice that point of interest features refer to image files, which we know are stored inside the default GeoServer distribution in the `demo_app/pics` path. So, we could provide a POI specific override that actually loads the images.

This is easy: just put the following template in the feature type folder, which in this case is `workspaces/topp/DS_poi/poi` (you should refer to your Internet visible server address instead of localhost, or its IP if you have fixed IPs):

```
<ul>
<#list features as feature>
```

Fig. 17.4: *Bulleted List Output*

```
  <li><b>Point of interest, "${feature.NAME.value}"</b>: <br/>
  <img src="http://localhost:8080/geoserver/popup_map/${feature.THUMBNAIL.value}"/>
  </li>
</#list>
</ul>
```

With this additional template, the output is:

As you can see, roads are still using the generic template, whilst POI is using its own custom template.

17.3.4 Advanced Formating

The `value` property of Feature attribute values are given by geoserver in `String` form, using a sensible default depending on the actual type of the attribute value. If you need to access the raw attribute value in order to apply a custom format (for example, to output `"Enabled"` or `"Disabled"` for a given boolean property, instead of the default `true/false`, you can just use the `rawValue` property instead of `value`. For example: `${attribute.rawValue?string("Enabled", "Disabled")}` instead of just `${attribute.value}`.

17.4 Paletted Images

GeoServer has the ability to output high quality 256 color images. This tutorial introduces you to the palette concepts, the various image generation options, and offers a quality/resource comparison of them in different situations.

Fig. 17.5: *Output with Thumbnail Image*

17.4.1 What are Paletted Images?

Some image formats, such as GIF or PNG, can use a palette, which is a table of (usually) 256 colors to allow for better compression. Basically, instead of representing each pixel with its full color triplet, which takes 24bits (plus eventual 8 more for transparency), they use a 8 bit index that represent the position inside the palette, and thus the color.

This allows for images that are 3-4 times smaller than the standard images, with the limitation that only 256 different colors can appear on the image itself. Depending of the actual map, this may be a very stringent limitation, visibly degrading the image quality, or it may be that the output cannot be told from a full color image. But for many maps one can easily find 256 representative colors.

In the latter case, the smaller footprint of paletted images is usually a big gain in both performance and costs, because more data can be served with the same internet connection, and the clients will obtain responses faster.

17.4.2 Formats and Antialiasing

Internet standards offer a variety of image formats, all having different strong and weak points. The three most common formats are:

- **JPEG**: a lossy format with tunable compression. JPEG is best suited for imagery layers, where the pixel color varies continuously from one pixel to the next one, and allows for the best compressed outputs. On the contrary, it's not suited to most vector layers, because even slight compression generates visible artifacts on uniform color areas.
- **PNG**: a non lossy format allowing for both full color and paletted. In full color images each pixel is encoded as a 24bits integer with full transparency information (so PNG images can be translucent), in paletted mode each pixel is an 8 bit index into a 256 color table (the palette). This format is best

suited to vector layers, especially in the paletted version. The full color version is sometimes referred as PNG24, the paletted version as PNG8.

- **GIF**: a non lossy format with a 256 color palette, best suited for vector layers. Does not support translucency, but allows for fully transparent pixels.

So, as it turns out, paletted images can be used with profit on vector data sets, either using the PNG8 or GIF formats.

Antialiasing plays a role too. Let's take a road layer, where each road is depicted by a solid gray line, 2 pixels thick. One may think this layer needs only 2 colors: the background one (eventually transparent) and gray. In fact, this is true only if no antialiasing is enabled. Antialiasing will smooth the borders of the line giving a softer, better looking shape, and it will do so by adding pixels with an intermediate color, thus increasing the number of colors that are needed to fully display the image.

The following zoom of an image shows antialiasing in action:

Fig. 17.6: *Antialiasing*

These output formats, if no other parameters are provided, do compute the optimal palette on the fly. As you'll see, this is an expensive process (CPU bound), but as you'll see, depending on the speed of the network connecting the server and the client, the extra cost can be ignored (especially if the bottleneck can be found in the network instead of the server CPU).

Optimal palette computation is anyways a repetitive work that can be done up front: a user can compute the optimal palette once, and tell GeoServer to use it. There are three ways to do so:

1. Use the internet safe palette, a standard palette built in into GeoServer, by appending `palette=safe` to the `GetMap` request.
2. Provide a palette by example. In this case, the user will generate an 256 color images using an external program (such as Photoshop), and then will save it into the `$GEOSERVER_DATA_DIR/palettes` directory. The sample file can be either in GIF or PNG format. If the file is named mypalette.gif or mypalette.png, the user will be able to refer it appending `palette=mypalette` to the GetMap request. GeoServer will load the palette from the file and use it.
3. Provide a palette file. The palette file must be in JASC-PAL format, and have a `.pal` extension. This file type can be generated by applications such as Paint Shop Pro and IrfanView, but also can be generated manually in a text editor. The process is just as before, but this time only the palette file will be stored into `$GEOSERVER_DATA_DIR/palettes`.

Note: GeoServer does not support palette files in Microsoft Palette format, despite having the same `.pal` file extension.

17.4.3 An Example with Vector Data

Enough theory, let's have a look at how to deal with paletted images in practice. We'll use the `tiger-ny` basemap to gather some numbers, and in particular the following map request:

And we'll change various parameters in order to play with formats and palettes. Here goes the sampler:

Parameters:`FORMAT=image/png` | Size: 257 KB | Map generation time: 0.3s

Fig. 17.7: *The standard PNG full color output*

Parameters:`FORMAT=image/png8` | Size: 60 KB | Map generation time: 0.6s

Fig. 17.8: *The PNG8 output*

Parameters:`FORMAT=image/png` | Size: 257 KB | Map generation time: 0.3s

Parameters:`FORMAT=image/png` & `palette=nyp` | Size: 56KB | Map generation time: 0.3s

The attachments include also the GIF outputs, whose size, appearance and generation time does not differ significantly from the PNG outputs.

As we can see, depending on the choice we have a variation on the image quality, size and generation time (which has been recorded using the FasterFox Firefox extension timer, with the browser sitting on the same box as the server). Using `palette=xxx` provides the best match in speed and size, thought using the built

Fig. 17.9: *PNG + internet safe palette*

Fig. 17.10: *PNG + 'custom palette <http://geoserver.org/download/attachments/1278244/nyp.pal?version=1>'_*

in internet safe palette altered the colors. Then again, the real gain can be seen only by assuming a certain connection speed between the server and the client, and adding the time required to move the image to the client. The following table provides some results:

Configuration	GT(s)	File size (kb)	TT 256kbit/s	TT 1MBit/s	TT 4MBit/s	TT 20MBit/s
tiger-ny-png	0,36	257	8,39	2,42	0,87	0,46
tyger-ny-png8	0,6	60	2,48	1,08	0,72	0,62
tiger-ny-png + safe palette	0,3	56	22,05	0,75	0,41	0,32
tiger-ny-png + custom palette	0,3	59	2,14	0,77	0,42	0,32

Legend:

- `GT`: map generation time on the same box
- `TT <speed>`: total time needed for a client to show the image, assuming an internet connection of the given speed. This time is a sum of of the image generation time and the transfer time, that is, `GT + sizeInKbytes * 8/ speedInKbits`.

As the table shows, the full color PNG image takes usually a lot more time than other formats, unless it's being served over a fast network (and even in this case, one should consider network congestion as well). The png8 output format proves to be a good choice if the connection is slow, whilst the extra work done in looking up an optimal palette always pays back in faster map delivery.

17.4.4 Generating the custom palette

The nyp.pal file has been generated using IrfanView, on Windows. The steps are simple:

- open the png 24 bit version of the image
- use Image/Decrease Color Depth and set 256 colors
- use Image/Palette/Export to save the palette

17.4.5 An example with raster data

To give you an example when paletted images may not fit the bill, let's consider the sf:dem coverage from the sample data, and repeat the same operation as before.

Parameters:`FORMAT=image/png` Size: 117 KB | Map generation time: 0.2s

Parameters:`FORMAT=image/jpeg` Size: 23KB | Map generation time: 0.12s

Parameters:`FORMAT=image/png8` Size: 60 KB | Map generation time: 0.5s

Parameters:`FORMAT=image/png` & `palette=dem-png8` Size: 48KB | Map generation time: 0.15s

Parameters:`FORMAT=image/png``& ``palette=safe` Size: 17KB | Map generation time: 0.15s

As the sample shows, the JPEG output has the same quality as the full color image, is generated faster and uses only 1/5 of its size. On the other hand, the version using the internet safe palette is fast and small, but the output is totally ruined. Everything considered, JPEG is the clear winner, sporting good quality, fast image generation and a size that's half of the best png output we can get.

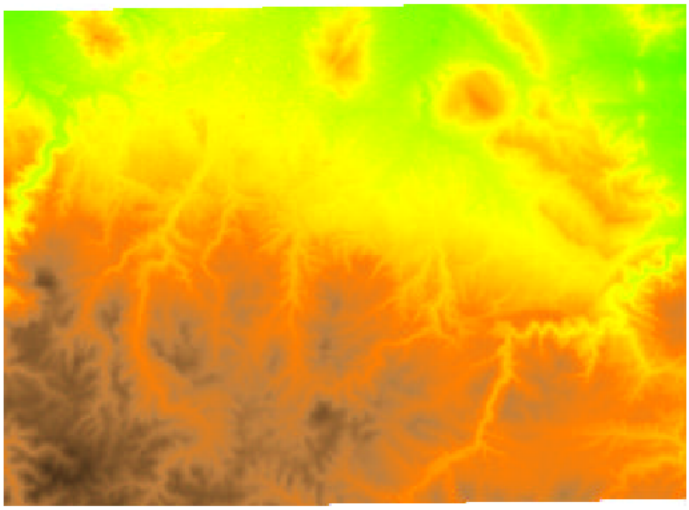


Fig. 17.11: *The standard PNG full color output*.

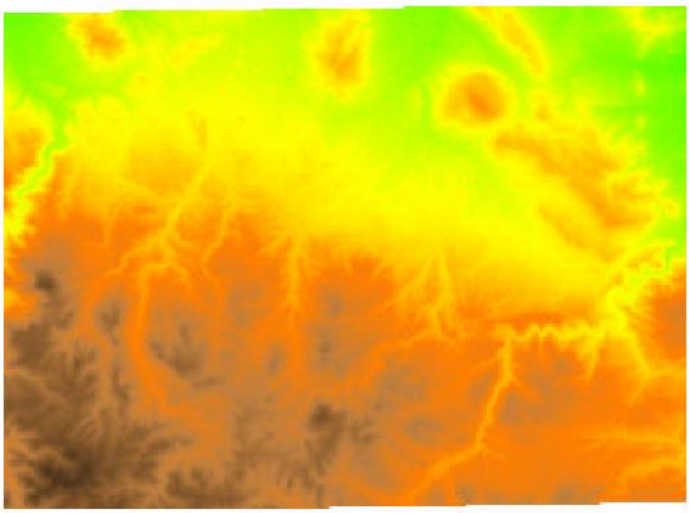


Fig. 17.12: *JPEG output*

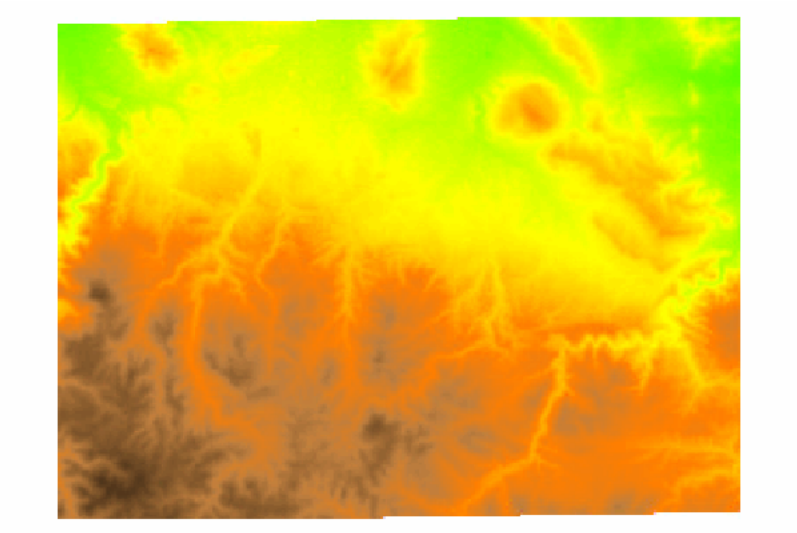


Fig. 17.13: *The PNG8 output.*

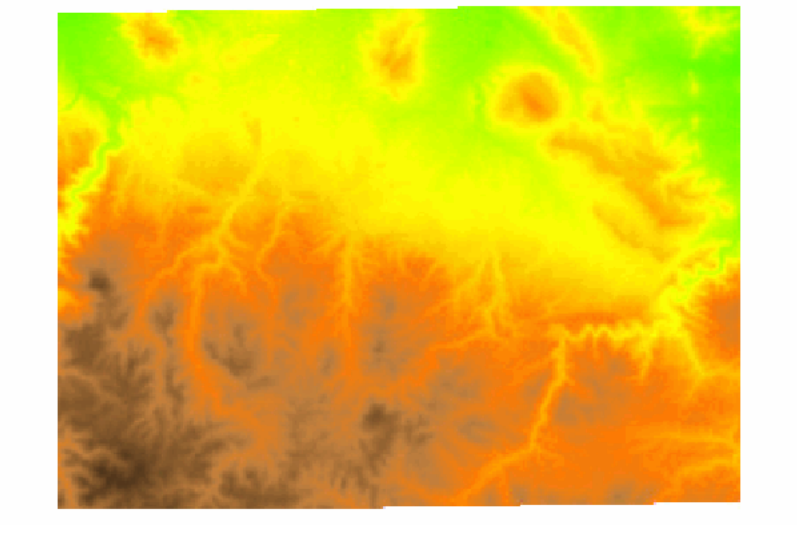


Fig. 17.14: *PNG + custom palette (using the png8 output as the palette).*

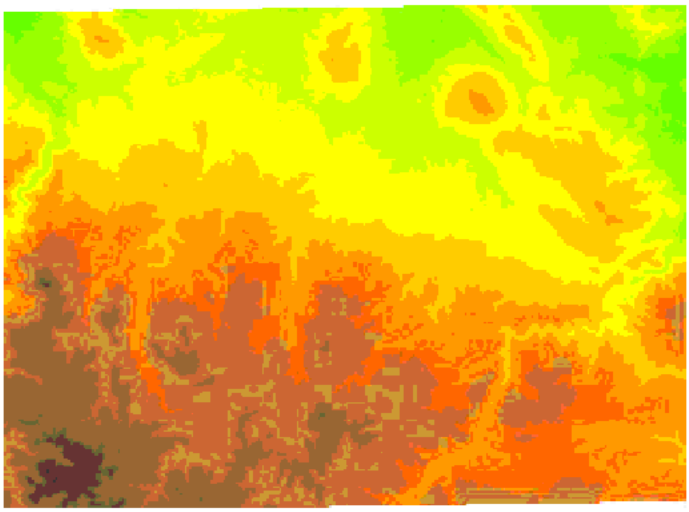


Fig. 17.15: *PNG + internet safe palette.*

17.5 Serving Static Files

You can place static files in the `www` subdirectory of the GeoServer *data directory*, and they will be served at `http:/myhost:8080/geoserver/www`. This means you can deploy HTML, images, or JavaScript, and have GeoServer serve them directly on the web.

This approach has some limitations:

- GeoServer can only serve files whose MIME type is recognized. If you get an HTTP 415 error, this is because GeoServer cannot determine a file's MIME type.
- This approach does not make use of accelerators such as the Tomcat APR library. If you have many static files to be served at high speed, you may wish to create your own web app to be deployed along with GeoServer or use a separate web server to serve the content.

17.6 WMS Reflector

17.6.1 Overview

Standard WMS requests can be quite long and verbose. For instance the following, which returns an OpenLayers application with an 800x600 image set to display the feature `topp:states`, with bounds set to the northwestern hemisphere by providing the appropriate bounding box:

```
http://localhost:8080/geoserver/wms?service=WMS&request=GetMap&version=1.1.1&
↪format=application/openlayers&width=800&height=600&srs=EPSG:4326&layers=topp:states&
↪styles=population&bbox=-180,0,0,90
```

Typing into a browser, or HTML editor, can be quite cumbersome and error prone. The WMS Reflector solves this problem nicely by using good default values for the options that you do not specify. Using the reflector one can shorten the above request to:

```
http://localhost:8080/geoserver/wms/reflect?format=application/openlayers&
↪layers=topp:states&width=800
```

This request only specifies that you want the reflector (wms/reflect) to return an OpenLayers application (format=application/openlayers), that you want it to display the feature "topp:states" (layers=topp:states) and that the width should be 800 pixels (width=800). However, this will not return the exact same value as above. Instead, the reflector will zoom to the bounds of the feature and return a map that is 800 pixels wide, but with the height adjusted to the aspect ratio of the feature.

17.6.2 Using the WMS Reflector

To use the WMS reflector all one must do is specify `wms/reflect?` as opposed to `wms?` in a request. The only mandatory parameter to a WMS reflector call is the *layers* parameter. As stated above the reflector fills in sensible defaults for the rest of the parameters. The following table lists all the defaults used:

request	getmap
service	wms
version	1.1.1
format	image/png
width	512
height	512 if width is not specified
srs	EPSG:4326
bbox	bounds of layer(s)

Any of these defaults can be overridden when specifying the request. The *styles* parameter is derived by using the default style as configured by GeoServer for each *layer* specified in the layers parameter.

Any parameter you send with a WMS request is also legitimate when requesting data from the reflector. Its strength is what it does with the parameters you do not specify, which is explored in the next section.

layers: This is the only mandatory parameter. It is a comma separated list of the layers you wish to include in your image or OpenLayers application.

format: The default output format is image/png. Alternatives include image/jpeg (good for raster backgrounds), image/png8 (8 bit colors, smaller files) and image/gif

width: Describes the width of the image, alternatively the size of the map in an OpenLayers. It defaults to 512 pixels and can be calculated based on the height and the aspect ratio of the bounding box.

height: Describes the height of the image, alternatively the map in an OpenLayers. It can be calculated based on the width and the aspect ratio of the bounding box.

bbox: The bounding box is automatically determined by taking the union of the bounds of the specified layers. In essence, it determines the extent of the map. By default, if you do not specify bbox, it will show you everything. If you have one layer of Los Angeles, and another of New York, it show you most of the United States. The bounding box, automatically set or specified, also determines the aspect ratio of the map. If you only specify one of width or height, the other will be determined based on the aspect ratio of the bounding box.

> **Warning:** If you specify height, width and bounding box there are zero degrees of freedom, and if the aspect ratios do not match your image will be warped.

styles: You can override the default styles by providing a comma separated list with the names of styles which must be known by the server.

srs: The spatial reference system (SRS) parameter is somewhat difficult. If not specified the WMS Reflector will use EPSG:4326 / WGS84. It will support the native SRS of the layers as well, provided all layers share the same one.

Example 1

Request the layer topp:states , it will come back with the default style (demographic), width (512 pixels) and height (adjusted to aspect ratio):

```
http://localhost:8080/geoserver/wms/reflect?layers=topp:states
```

Example 2

Request the layers topp:states and sf:restricted, it will come back with the default styles, and the specified width (640 pixels) and the height automatically adjusted to the aspect ratio:

```
http://localhost:8080/geoserver/wms/reflect?layers=topp:states,sf:restricted&width=640
```

Example 3

In the example above the sf:restricted layer is very difficult to see, because it is so small compared to the United States. To give the user a chance to get a better view, if they choose, we can return an OpenLayers application instead. Zoom in on South Dakota (SD) to see the restricted areas:

```
http://localhost:8080/geoserver/wms/reflect?format=application/openlayers&
↪layers=topp:states,sf:restricted&width=640
```

Example 4

Now, if you mainly want to show the restricted layer, but also provide the context, you can set the bounding box for the the request. The easiest way to obtain the coordinates is to use the application in example three and the coordinates at the bottom right of the map. The coordinates displayed in OpenLayers are x , y , the reflector service expects to be given bbox=minx,miny,maxx,maxy . Make sure it contains no whitespaces and users a period (".") as the decimal separator. In our case, it will be bbox=-103.929,44.375,-103.633,44.500

```
http://localhost:8080/geoserver/wms/reflect?format=application/openlayers&
↪layers=topp:states,sf:restricted&width=640&bbox=-103.929,44.375,-103.633,44.500
```

17.6.3 Outputting to a Webpage

Say you have a webpage and you wish to include a picture that is 400 pixels wide and that shows the layer `topp:states`, on this page.

```
<img src="http://localhost:8080/geoserver/wms/reflect?layers=topp:states&width=400" />
```

If you want the page to render in the browser before GeoServer is done, you should specify the height and width of the picture. You could just pick any approximate value, but it may be a good idea to look at the generated image first and then use those values. In the case of the layer above, the height becomes 169 pixels, so we can specify that as an attribute in the <img> tag:

```
<img src="http://localhost:8080/geoserver/wms/reflect?layers=topp:states&width=400"␣
↪height="169" width="400"/>
```

If you are worried that the bounds of the layer may change, so that the height changes relative to the width, you may also want to specify the height in the URL to the reflector. This ensures the layer will always be centered and fit on the 400x169 canvas.

The reflector can also create a simple instance of OpenLayers that shows the layers you specify in your request. One possible application is to turn the image above into a link that refers to the OpenLayers instance for the same feature, which is especially handy if you think a minority of your users will want to take closer look. To link to this JavaScript application, you need to specify the output format of the reflector: `format=application/OpenLayers`:

```
http://localhost:8080/geoserver/wms/reflect?format=application/openlayers&width=400
```

The image above then becomes

```
<a href="http://localhost:8080/geoserver/wms/reflect?format=application/openlayers&
↪layers=topp:states">
<img src="http://localhost:8080/geoserver/wms/reflect?layers=topp:states&width=400"␣
↪height="169" width="400" />
</a>
```

(The a-tags are on separate lines for clarity, they will in fact result in a space in front and after the image).

17.6.4 OpenLayers in an iframe

Many people do not like iframes, and for good reasons, but they may be appropriate in this case. The following example will run OpenLayers in an iframe.

```
<iframe src ="http://localhost:8080/geoserver/wms/reflect?format=application/
↪openlayers&layers=topp:states" width="100%">
</iframe>
```

Alternatively, you can open OpenLayers in a separate webpage and choose "View Source code" in your browser. By copying the HTML you can insert the OpenLayers client in your own page without using an iframe.

17.7 WMS Animator

17.7.1 Overview

Standard WMS can generate static maps only. There is a number of use cases in which generating an animation is of interest. An obvious case is time-based animation. Other uses include elevation-based animation, varying the values of SQL View or SLD substitution parameters, or the changing the extent of the generated map to produce the appearance of a moving viewport.

This capability is provided by the **WMS Animator**. The WMS Animator works in a similar way to the WMS Reflector. It uses a provided partial WMS request as a template, and the **animator parameters** are used to generate and execute a sequence of complete requests. The rendered map images are combined into a single output image (in a format that supports multi-frame images).

The Animator is invoked by using the `wms/animate` request path. Any WMS paramaters can be animated, including nested ones such as *SLD environment variables*. To define the appearance of the animation additional parameters are provided:

- **aparam** specifies the name of the parameter that will be changed in the request for each frame. This can be any WMS parameter such as `layers`, `cql_filter`, `bbox`, `style` and so on. Nested parameters (such as required by the `format_options`, `env` and `view_params` parameters), are supported using the syntax of `param:name` (for example, `view_params:year`).
- **avalues** is a comma-separated list of the values the animation parameter has for each frame. If a value contain commas these must be escaped using a backslash. (For instance, this occurs when providing BBOX values.)

The Animator parses the input values and uses string replacement to generate the sequence of WMS requests to be executed. Each generated request is executed to produce one frame. It is up to the caller to ensure the provided animation parameters result in valid WMS requests.

For example, to generate an animation of a layer with the viewport scrolling towards the east, the WMS BBOX parameter is given the series of values `-90,40,-60,70`, `-80,40,-60,70` and `-70,40,-50,70` (note the escaping of the commas in the BBOX values):

```
http://localhost:8080/geoserver/wms/animate
?layers=topp:states
&aparam=bbox
&avalues=-90\,40\,-60\,70,-80\,40\,-60\,70,-70\,40\,-50\,70
```

For an example of nested parameters, assume the existence of a style named `selection` using an SLD variable `color`. The following request creates an animated map where the selection color changes between red, green and blue:

```
http://localhost:8080/geoserver/wms/animate
?layers=topp:states,topp:states
&styles=polygon,selection
&aparam=env:color
&avalues=FF0000,00FF00,0000FF
```

17.7.2 Using the WMS Animator

To invoke the WMS Animator specify the path `wms/animate` instead of `wms` in a GetMap request.

Every Animator request must specify the `layers`, `aparam` and `avalues` parameters. Any other valid WMS parameters may be used in the request as well. If any necessary parameters are omitted, the Animator provides sensible default values for them. The following defaults are used:

Parameter	Default Value
`request`	`getmap`
`service`	`wms`
`version`	`1.1.1`
`format`	`image/png`
`width`	`512`
`height`	`512` if width is not specified
`srs`	`EPSG:4326`, or SRS common to all layers
`bbox`	bounds of specified layer(s)
`styles`	default styles configured for specified layer(s)

Further details of these parameters are:

layers: This is the only mandatory standard parameter. It is a comma-separated list of the layers to be included in the output map.

format: The default output format is `image/png`. Supported values are `image/jpeg` (suitable for raster backgrounds), `image/png8` (8-bit colors, smaller files) and `image/gif`

> **Warning:** In order to produce an actual animated image the format must support animation. At this time the only one provide in GeoServer is **image/gif;subtype=animated**

width: Describes the width of the image. It defaults to 512 pixels, and can be calculated based on the specified height and the aspect ratio of the bounding box.

height: Describes the height of the image. It can be calculated based on the specified width and the aspect ratio of the bounding box.

bbox: Specifies the extent of the map frame. The default bounding box is determined by taking the union of the bounds of the specified layers. (For example, if one layer shows Los Angeles and another shows New York, the default map shows most of the United States. The bounding box also determines the aspect ratio of the map. If only one of `width` or `height` is specified, the other is determined based on the aspect ratio of the bounding box.

styles: The default value is the default styles configured in GeoServer for the layers specified in the `layers` parameter. This can be overridden by providing a comma-separated list of style names (which must be known to the server).

srs: If all layers share the same SRS, this is used as the default value. Otherwise, the default value is EPSG:4326 (WGS84).

Animation Options

The Animator provides options to control looping and frame speed. These are specified using the `format_options` *WMS parameter*. The available options are:

Option	Description
`gif_loop_continuously`	If `true` the animation will loop continuously. The default is `false`.
`gif_frames_delay`	Specifies the frame delay in milliseconds. The default is 1000 ms.
`gif_disposal`	Specifies what to do with the previous GIF frame once a new frame is displayed. Valid values are `none`, `doNotDispose`, `backgroundColor` and `previous`. The default is `none`.

Example 1

Requests the layer `topp:states`, using the default style (`demographic`), width (512 pixels) and height (adjusted to aspect ratio). The `aparam=bbox` parameter specifies that the output animation has two frames, one using a whole-world extent and the other with the extent of the USA. This gives the effect of zooming in:

```
http://localhost:8080/geoserver/wms/animate
?layers=topp:states
&format=image/gif;subtype=animated
&aparam=bbox
&avalues=-180\,-90\,180\,90,-125\,25\,-67\,50
```

Example 2

Requests the layers `topp:states` and `sf:restricted`, using `format_options=gif_loop_continuosly:true` to request an infinite loop animation. The output map uses the default styles, the specified width (640 pixels), and the height automatically adjusted to the aspect ratio:

```
http://localhost:8080/geoserver/wms/animate
?layers=topp:states,sf:restricted
&format=image/gif;subtype=animated
&aparam=bbox
&avalues=-180\,-90\,180\,90,-125\,25\,-67\,50
&format_options=gif_loop_continuosly:true
&width=640
```

Example 3

The following request uses the `format_options` of `gif_loop_continuosly:true` and `gif_frames_delay:10` to rotate the map image fast and continuously:

```
http://localhost:8080/geoserver/wms/animate
?layers=topp:states,sf:restricted
&format=image/gif;subtype=animated
&aparam=angle
&avalues=0,45,90,135,180,225,270,365
&format_options=gif_loop_continuosly:true;gif_frames_delay:10
&width=640
```

17.7.3 Displaying frame parameters as decorations

It is possible to decorate each frame image with the `avalue` parameter value that generated it using the *WMS Decorations* `text` decoration. The current animation parameter value can be accessed via the `avalue` environment variable. (This environment variable can also be used in *Variable substitution in SLD*.)

Here is an example that uses a decoration showing the frame parameter value:

```
http://localhost:8080/geoserver/wms/animate
?layers=topp%3Aworld
&aparam=time
&avalues=2004-01-01T00:00:00.000Z,2004-02-01T00:00:00.000Z
&format=image/gif;subtype=animated
&format_options=layout:message
```

It uses the following decoration layout, located in `layouts/message.xml`:

```
<layout>
  <decoration type="text" affinity="bottom,right" offset="6,6">
    <option name="message" value="${avalue}"/>
    <option name="font-size" value="12"/>
    <option name="font-family" value="Arial"/>
    <option name="halo-radius" value="2"/>
  </decoration>
</layout>
```

17.7.4 Specifying WMS Animator default behaviour

The GeoServer Adinistrator GUI allows specifying some limits and default options for the WMS Animator. The settings are made on the *Services* > *WMS* config screen as shown below:

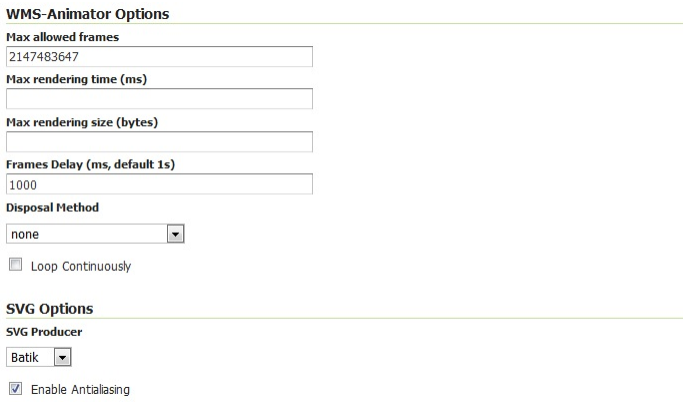


Fig. 17.16: *WMS Animator default settings*

The first three options set server limits on the animation output. It is possible to set the **maximum number of frames** an animation can contain, the **maximum rendering time** to produce an animation and the **maximum size** of the whole animation.

The default animation **frame delay** (expressed in ms) **looping behaviour** and **disposal method** can be set as well. These values can be overridden by using the `format_options` parameter as described above.

17.8 CQL and ECQL

CQL (Common Query Language) is a query language created by the OGC for the Catalogue Web Services specification. Unlike the XML-based Filter Encoding language, CQL is written using a familiar text-based syntax. It is thus more readable and better-suited for manual authoring.

However, CQL has some limitations. For example it cannot encode id filters, and it requires an attribute to be on the left side of any comparison operator. For this reason, GeoServer provides an extended version of CQL called ECQL. ECQL removes the limitations of CQL, providing a more flexible language with stronger similarities with SQL.

GeoServer supports the use of both CQL and ECQL in WMS and WFS requests, as well as in GeoServer's SLD *dynamic symbolizers*. Whenever the documentation refers to CQL, ECQL syntax can be used as well (and if not, please report that as a bug!).

This tutorial introduces the CQL/ECQL language by example. For a full reference, refer to the *ECQL Reference*.

17.8.1 Getting started

The following examples use the `topp:states` sample layer shipped with GeoServer. They demonstrate how CQL filters work by using the WMS *CQL_FILTER vendor parameter* to alter the data displayed by WMS requests. The easiest way to follow the tutorial is to open the GeoServer Map Preview for the `topp:states` layer. Click on the *Options* button at the top of the map preview to open the advanced options toolbar. The example filters can be entered in the *Filter: CQL* box.

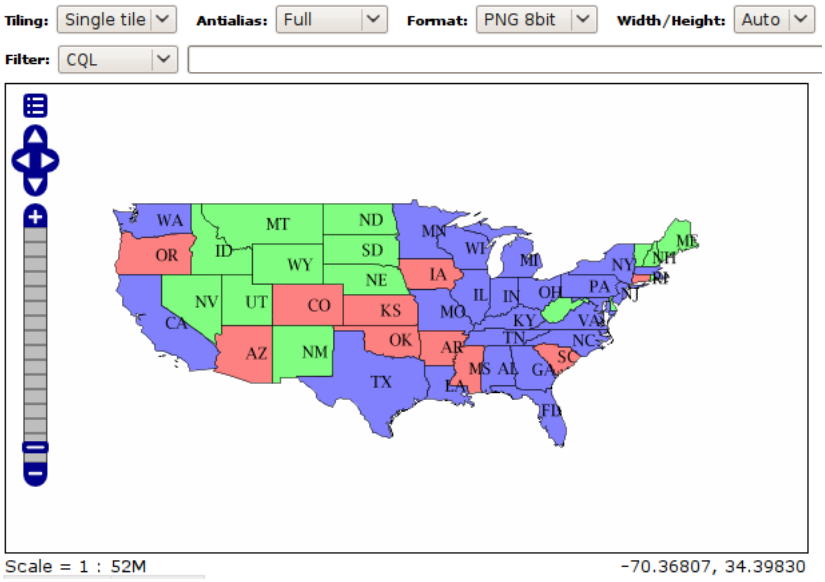


Fig. 17.17: *topp:states preview with advanced toolbar open.*

The attributes used in the filter examples are those included in the layer. For example, the following are the attribute names and values for the Colorado feature:

Attribute	**states.6**
STATE_NAME	Colorado
STATE_FIPS	08
SUB_REGION	Mtn
STATE_ABBR	CO
LAND_KM	268659.501
WATER_KM	960.364
PERSONS	3294394.0
FAMILIES	854214.0
HOUSHOLD	1282489.0
MALE	1631295.0
FEMALE	1663099.0
WORKERS	1233023.0
DRVALONE	1216639.0
CARPOOL	210274.0
PUBTRANS	46983.0
EMPLOYED	1633281.0
UNEMPLOY	99438.0
SERVICE	421079.0
MANUAL	181760.0
P_MALE	0.495
P_FEMALE	0.505
SAMP_POP	512677.0

17.8.2 Simple comparisons

Let's get started with a simple example. In CQL arithmetic and comparisons are expressed using plain text. The filter `PERSONS > 15000000` will select states that have more than 15 million inhabitants:

Fig. 17.18: *PERSONS > 15000000*

The full list of comparison operators is: =, <>, >, >=, <, <=.

To select a range of values the BETWEEN operator can be used: `PERSONS BETWEEN 1000000 AND 3000000`:

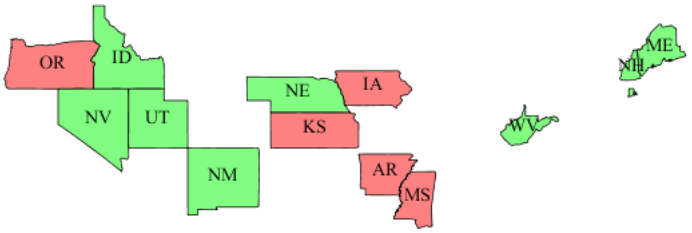


Fig. 17.19: *PERSONS BETWEEN 1000000 AND 3000000*

Comparison operators also support text values. For instance, to select only the state of California, the filter is `STATE_NAME = 'California'`. More general text comparisons can be made using the `LIKE` operator. `STATE_NAME LIKE 'N%'` will extract all states starting with an "N":

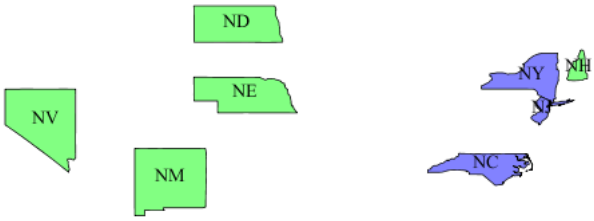


Fig. 17.20: *STATE_NAME LIKE 'N%'*

It is also possible to compare two attributes with each other. `MALE > FEMALE` selects the states in which the male population surpasses the female one (a rare occurrence):

Fig. 17.21: *MALE > FEMALE*

Arithmetic expressions can be computed using the `+`, `-`, `*`, `/` operators. The filter `UNEMPLOY / (EMPLOYED + UNEMPLOY) > 0.07` selects all states whose unemployment ratio is above 7% (remember the sample data is very old, so don't draw any conclusion from the results!)

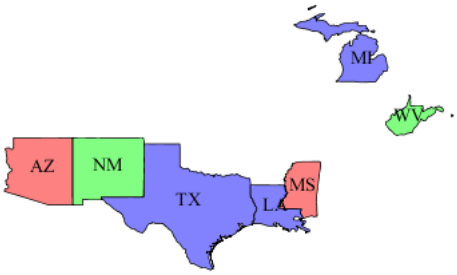


Fig. 17.22: *UNEMPLOY / (EMPLOYED + UNEMPLOY) > 0.07*

17.8.3 Id and list comparisons

If we want to extract only the states with specific feature ids we can use the `IN` operator without specifying any attribute, as in `IN ('states.1', 'states.12')`:

Fig. 17.23: *IN ('states.1', 'states.12')*

If instead we want to extract the states whose name is in a given list we can use the `IN` operator specifying an attribute name, as in `STATE_NAME IN ('New York', 'California', 'Montana', 'Texas')`:

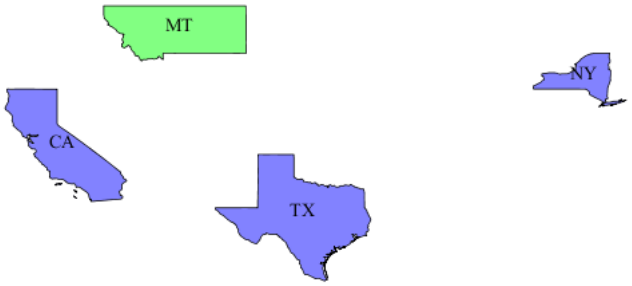


Fig. 17.24: *STATE_NAME IN ('New York', 'California', 'Montana', 'Texas')*

17.8.4 Filter functions

CQL/ECQL can use any of the *filter functions* available in GeoServer. This greatly increases the power of CQL expressions.

For example, suppose we want to find all states whose name contains an "m", regardless of letter case. We can use the `strToLowerCase` to turn all the state names to lowercase and then use a like comparison: `strToLowerCase(STATE_NAME) like '%m%'`:

Fig. 17.25: *strToLowerCase(STATE_NAME) like '%m%'*

17.8.5 Geometric filters

CQL provides a full set of geometric filter capabilities. Say, for example, you want to display only the states that intersect the (-90,40,-60,45) bounding box. The filter will be `BBOX(the_geom, -90, 40, -60, 45)`

Fig. 17.26: *BBOX(the_geom, -90, 40, -60, 45)*

Conversely, you can select the states that do *not* intersect the bounding box with the filter: `DISJOINT(the_geom, POLYGON((-90 40, -90 45, -60 45, -60 40, -90 40)))`:

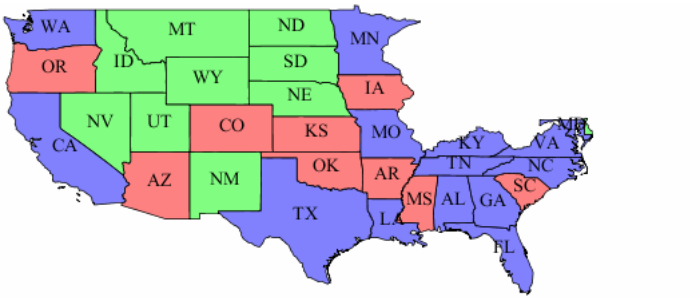


Fig. 17.27: *DISJOINT(the_geom, POLYGON((-90 40, -90 45, -60 45, -60 40, -90 40)))*

The full list of geometric predicates is: `EQUALS`, `DISJOINT`, `INTERSECTS`, `TOUCHES`, `CROSSES`, `WITHIN`, `CONTAINS`, `OVERLAPS`, `RELATE`, `DWITHIN`, `BEYOND`.

17.9 Using the ImageMosaic plugin for raster time-series data

17.9.1 Introduction

This step-by-step tutorial describes how to build a time-series coverage using the ImageMosaic plugin. The ImageMosaic plugin allows the creation of a time-series layer of a raster dataset. The single images are held in a queryable structure to allow access to a specific dataset with a temporal filter.

This tutorial assumes knowledge of the concepts explained in *ImageMosaic* section.

This tutorial contains four sections:

- The first section, **Configuration**, describes the configuration files needed to set up an ImageMosaic store from GeoServer.
- The second section, **Configuration examples**, providing examples of the configuration files needed.
- The last two sections, **Coverage based on filestore** and **Coverage based on database** describe, once the previous configurations steps are done, how to create and configure an ImageMosaic store using the GeoServer GUI.

The dataset used in the tutorial can be downloaded `Here`. It contains 3 image files and a .sld file representing a style needed for correctly render the images.

17.9.2 Configuration

To support time-series layers, GeoServer needs to be run in a web container that has the **timezone properly configured**. To set the time zone to be Coordinated Universal Time (UTC), add this switch when launching the java process:

```
-Duser.timezone=GMT
```

If using a shapefile as the mosaic index store (see next section), another java process option is needed to enable support for timestamps in shapefile stores:

```
-Dorg.geotools.shapefile.datetime=true
```

Note: Support for timestamp is not part of the DBF standard (used in shapefile for attributes). The DBF standard only supports Date, and only few applications understand it. As long as shapefiles are only used for GeoServer input that is not a problem, but the above setting will cause issues if you have WFS enabled and users also download shapefiles as GetFeature output: if the feature type extracted has timestamps, then the generated shapefile will have as well, making it difficult to use the generated shapefile in desktop applications. As a rule of thumb, if you also need WFS support it is advisable to use an external store (PostGIS, Oracle) instead of shapefile. Of course, if all that's needed is a date, using shapefile as an index without the above property is fine as well.

In order to load a new CoverageStore from the GeoServer GUI two steps are needed:

1. Create a new directory in which you store all the raster files (the mosaic granules) and three configuration files. This directory represents the **MOSAIC_DIR**.
2. Install and setup a DBMS instance, this DB is that one where the mosaic indexes will be stored.

MOSAIC_DIR and the Configuration Files

The user can name and place the **MOSAIC_DIR** as and where they want.

The **MOSAIC_DIR** contains all mosaic granules files and the 3 needed configuration files. The files are in `.properties` format.

Note: Every tif file must follow the same naming convention. In this tutorial will be used {coverage-name}_{timestamp}.tif

In a properties file you specify your properties in a key-value manner: e.g. *myproperty=myvalue*

The configuration files needed are:

1. **datastore.properties**: contains all relevant information responsible for connecting to the database in which the spatial indexes of the mosaic will be stored
2. **indexer.properties**: specifies the name of the time-variant attribute, the elevation attribute and the type of the attributes
3. **timeregex.properties**: specifies the regular expression used to extract the time information from the filename.

All the configuration files must be placed in the root of the **MOSAIC_DIR**. The granule images could be placed also in **MOSAIC_DIR** subfolders.

Please note that **datastore.properties** isn't mandatory. The plugin provides two possibilities to access to time series data:

- **Using a shapefile** in order to store the granules indexes. That's the default behavior without providing the *datastore.properties* file.
- **Using a DBMS**, which maps the timestamp to the corresponding raster source. The former uses the **time** attribute for access to the granule from the mapping table.

For production installation is strong recommended the usage of a DBMS instead of shapefile in order to improve performances.

Otherwise the usage of a shapefile could be useful in development and test environments due to less configurations are needed.

datastore.properties

Here is shown an example of datastore.properties suitable for Postgis.

Parameter	Mandatory	Description
SPI	**Y**	The factory class used for the datastore e.g. org.geotools.data.postgis.PostgisNGDataStoreFactory
host	**Y**	The host name of the database.
port	**Y**	The port of the database
database	**Y**	The name/instance of the database.
schema	**Y**	The name of the database schema.
user	**Y**	The database user.
passwd	**Y**	The database password.
Loose bbox	**N** default 'false'	Boolean value to specify if loosing the bounding box.
Estimated extend	**N** default 'true'	Boolean values to specify if the extent of the data should be estimated.
validate connections	**N** default 'true'	Boolean value to specify if the connection should be validated.
Connection timeout	**N** default '20'	Specifies the timeout in minutes.
preparedStatements	**N** default 'false'	Boolean flag that specifies if for the database queries prepared statements should be used. This improves performance, because the database query parser has to parse the query only once

Note: The first 8 parameters are valid for each DBMS used, the last 4 may vary from different DBMS. for more information see GeoTools JDBC documentation

indexer.properties

Parameter	Mandatory	Description
TimeAttribute	N	Specifies the name of the time-variant attribute
ElevationAttribute	N	Specifies the name of the elevation attribute.
Schema	Y	A comma separated sequence that describes the mapping between attribute and the data type.
PropertyCollectors	Y	Specifies the extractor classes.

Warning: **TimeAttribute** is not a mandatory param but for the purpose of this tutorial is needed.

timeregex.properties

Parameter	Mandatory	Description
regex	Y	Specifies the pattern used for extracting the information from the file

After this you can create a new ImageMosaic datastore.

Install and setup a DBMS instance

First of all, note that the usage of a DBMS to store the mosaic indexes **is not mandatory**. If the user does not create a *datastore.properties* file in the MOSAIC_DIR, then the plugin will use a **shapefile**. The shapefile will be created in the MOSAIC_DIR.

Anyway, especially for large dataset, **the usage of a DBMS is strongly recommended**. The ImageMosaic plugin supports all the most used DBMS.

The configuration needed are the basics: create a new empty DB with geospatial extensions, a new schema and configure the user with W/R grants.

If the user wants to avoid to manually create the DB, it will be automatically generated by the ImageMosaic plugin taking the information defined inside the *datastore.properties* file.

Note: In case of automatic DB creation with PostgreSQL the user must check the PostgreSQL and PostGIS versions: if the first is lower than 9.1 and the second is lower than 2.0, the user have to add the following string of code inside the *datastore.properties* file :

```
create\ database\ params=WITH\ TEMPLATE\=template_postgis
```

(Specifying the proper PostGIS template... in this example: *template_postgis*).

This tutorial shows use of PostgreSQL 9.1 together with PostGIS 2.0.

17.9.3 Configuration examples

As example is used a set of data that represents hydrological data in a certain area in South Tyrol, a region in northern Italy. The origin data was converted from asc format to TIFF using the GDAL **gdal translate** utility.

For this running example we will create a layer named snow.

As mentioned before the files could located in any part of the file system.

In this tutorial the chosen MOSAIC_DIR directory is called `hydroalp` and is placed under the root of the GEOSERVER_DATA_DIR.

Configure the MOSAIC_DIR:

This part showsn an entire MOSAIC_DIR configuration.

datastore.properties:

```
SPI=org.geotools.data.postgis.PostgisNGDataStoreFactory
host=localhost
port=5432
database=db
schema=public
user=dbuser
passwd=dbpasswd
```

```
Loose\ bbox=true
Estimated\ extends=false
validate\ connections=true
Connection\ timeout=10
preparedStatements=true
```

Note: In case of a missing datastore.properties file a shape file is created to hold the indexes.

Granules Naming Convention

Here an example of the granules naming that satisfies the rule shown before:

```
$ls hydroalp/snow/*.tif

snow/snow_20091001.tif
snow/snow_20091101.tif
snow/snow_20091201.tif
snow/snow_20100101.tif
snow/snow_20100201.tif
snow/snow_20100301.tif
snow/snow_20100401.tif
snow/snow_20100501.tif
snow/snow_20100601.tif
snow/snow_20100701.tif
snow/snow_20100801.tif
snow/snow_20100901.tif
```

timeregex.properties:

In the timeregex property file you specify the pattern describing the date(time) part of the file names. In this example it consists simply of 8 digits as specified below.

```
regex=[0-9]{8}
```

indexer.properties:

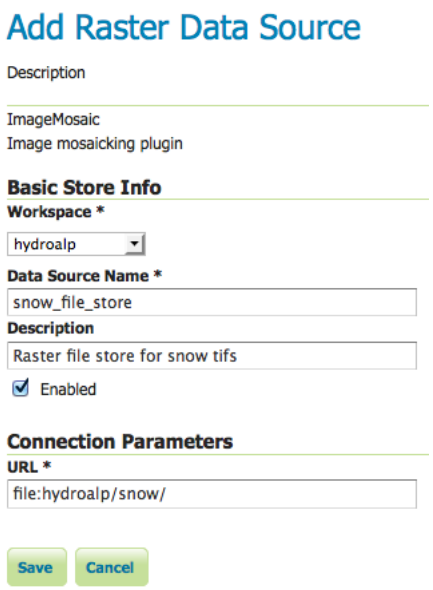
Here the user can specify the information that GeoServer uses to create the index table in the database. In this example, the time values are stored in the column ingestion.

```
TimeAttribute=ingestion
ElevationAttribute=elevation
Schema=*the_geom:Polygon,location:String,ingestion:java.util.Date,elevation:Integer
PropertyCollectors=TimestampFileNameExtractorSPI[timeregex](ingestion)
```

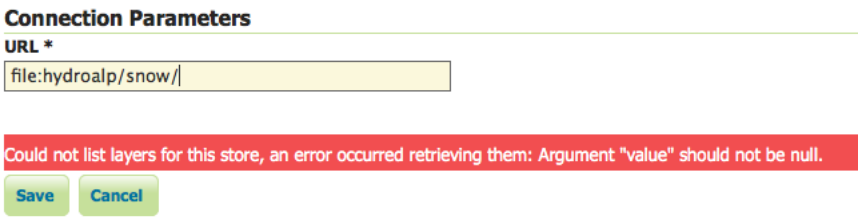
17.9.4 Create and Publish an ImageMosaic store:

Step 1: Create new ImageMosaic data store

We create a new data store of type raster data and choose ImageMosaic.

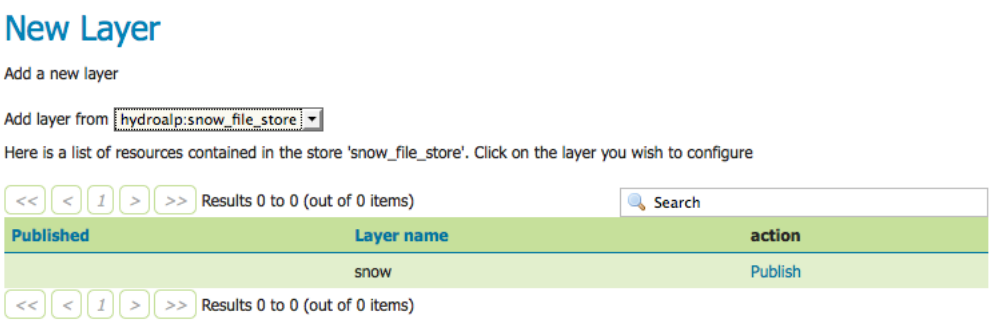


Note: Be aware that GeoServer creates a table which is identical with the name of your layer. If the table already exists, it will not be dropped from the DB and the following error message will appear. The same message wwill appear if the generated property file already exists in the directory or there are incorrect connection parameters in datastore.properties file.



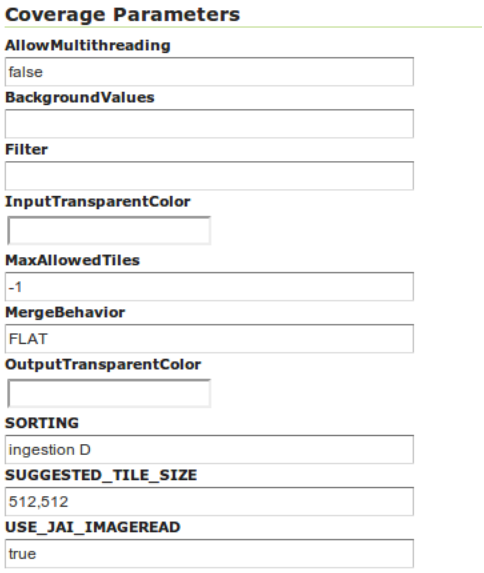
Step 2: Specify layer

We specify the directory that contains the property and TIFF files (path must end with a slash) and add the layer.



Step 3: Set coverage parameters

The relevant parameters are AllowMultithreading and USE_JAI_IMAGEREAD. Do not forget to specify the background value according to your the value in your tif file. If you want to control which granule is displayed when a number of images match the time period specified then set the SORTING parameter to the variable name you want to use for sorting followed by a space and either D or A for descending or ascending. Descending values will mean that the latest image in a series that occurs in the time period requested is shown.



Remember that for display correctly the images contained in the provided dataset a custom style is needed.

Set as default style the *snow_style.sld* contained in the dataset archive.

More information about raster styling can be found in chapter *Rasters*

Step 4: Set temporal properties

In the Dimensions tab you can specify how the time attributes are represented.

By enabling the Time or Elevation checkbox you can specify the how these dimensions will be presented. In this example, queries are only over the time attribute.

Below is shown a snippet of the Capabilities document for each presentation case:

Setting the presentation to **List**, all mosaic times are listed:

```
<Dimension name="time" default="current" units="ISO8601">
        2009-10-01T00:00:00.000Z,2009-11-01T00:00:00.000Z,2009-12-01T00:00:00.000Z,
↪2010-01-01T00:00:00.000Z,2010-02-01T00:00:00.000Z,2010-03-01T00:00:00.000Z,2010-04-
↪01T00:00:00.000Z,2010-05-01T00:00:00.000Z,2010-06-01T00:00:00.000Z,2010-07-
↪01T00:00:00.000Z,2010-08-01T00:00:00.000Z,2010-09-01T00:00:00.000Z,2010-10-
↪01T00:00:00.000Z,2010-11-01T00:00:00.000Z,2010-12-01T00:00:00.000Z,2011-01-
↪01T00:00:00.000Z,2011-02-01T00:00:00.000Z,2011-03-01T00:00:00.000Z,2011-04-
↪01T00:00:00.000Z,2011-05-01T00:00:00.000Z,2011-06-01T00:00:00.000Z,2011-07-
↪01T00:00:00.000Z,2011-08-01T00:00:00.000Z,2011-09-01T00:00:00.000Z
</Dimension>
```

Setting the presentation to **Continuous interval** only the start, end and interval extent times are listed:

```
<Dimension name="time" default="current" units="ISO8601">
        2009-10-01T00:00:00.000Z/2011-09-01T00:00:00.000Z/P1Y11MT10H
</Dimension>
```

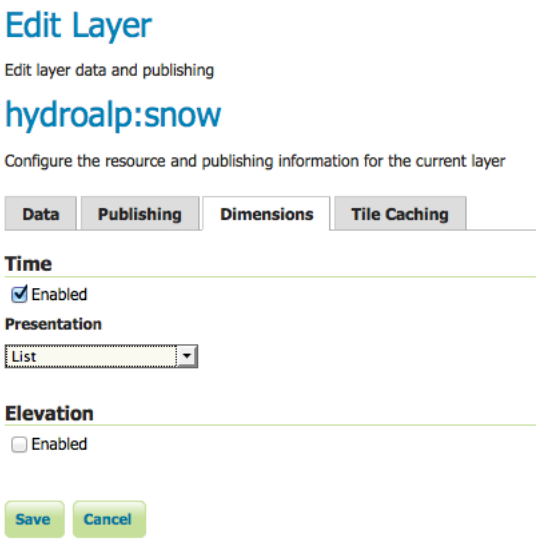
Setting the presentation to **Interval and resolutions** gives to user the possibility to specify the resolutions of the interval:

```
<Dimension name="time" default="current" units="ISO8601">
        2009-10-01T00:00:00.000Z/2011-09-01T00:00:00.000Z/P1DT12H
</Dimension>
```

In this case the resolution is set to 1.5 days.

Note: To visualize the GetCapabilities document, go to the GeoServer homepage, and click on the WMS 1.3.0 link under the tab labeled **Service Capabilities**.

For this tutorial the Presentation attribute is set to **List**



After this steps the new layer is available in GeoServer. GeoServer will create a property file in the source directory. GeoServer will either create a shapefile for the mosaic indexes, or will create a table on the database (named the same as the layer name) for the index.

Generated property file:

```
#-Automagically created from GeoTools-
#Sat Oct 13 10:47:08 CEST 2012
Levels=100.0,100.0
Heterogeneous=false
ElevationAttribute=elevation
TimeAttribute=ingestion
AbsolutePath=false
Name=snow
Caching=false
```

```
ExpandToRGB=false
LocationAttribute=location
SuggestedSPI=it.geosolutions.imageioimpl.plugins.tiff.TIFFImageReaderSpi
LevelsNum=1
```

Note: The parameter **Caching=false** is important to allow the user is to update manually the mosaic, by adding to and removing granules from MOSAIC_DIR and updating the appropriate database entry.

Generated table:

	fid [PK] serial	the_geom geometry	location character varying(255)	ingestion timestamp wi	elevation integer
1	1	01030000	snow_20091001.tif	2009-10-01	
2	2	01030000	snow_20091101.tif	2009-11-01	
3	3	01030000	snow_20091201.tif	2009-12-01	
4	4	01030000	snow_20100101.tif	2010-01-01	
5	5	01030000	snow_20100201.tif	2010-02-01	
6	6	01030000	snow_20100301.tif	2010-03-01	
7	7	01030000	snow_20100401.tif	2010-04-01	
8	8	01030000	snow_20100501.tif	2010-05-01	
9	9	01030000	snow_20100601.tif	2010-06-01	
10	10	01030000	snow_20100701.tif	2010-07-01	
11	11	01030000	snow_20100801.tif	2010-08-01	
12	12	01030000	snow_20100901.tif	2010-09-01	

Note: The user must create manually the index on the table in order to speed up the search by attribute.

Step 5: query layer on timestamp:

In order to display a snapshot of the map at a specific time instant you have to pass in the request an additional time parameter with a specific notation **&time=** < **pattern** > where you pass a value that corresponds to them in the filestore. The only thing is the pattern of the time value is slightly different.

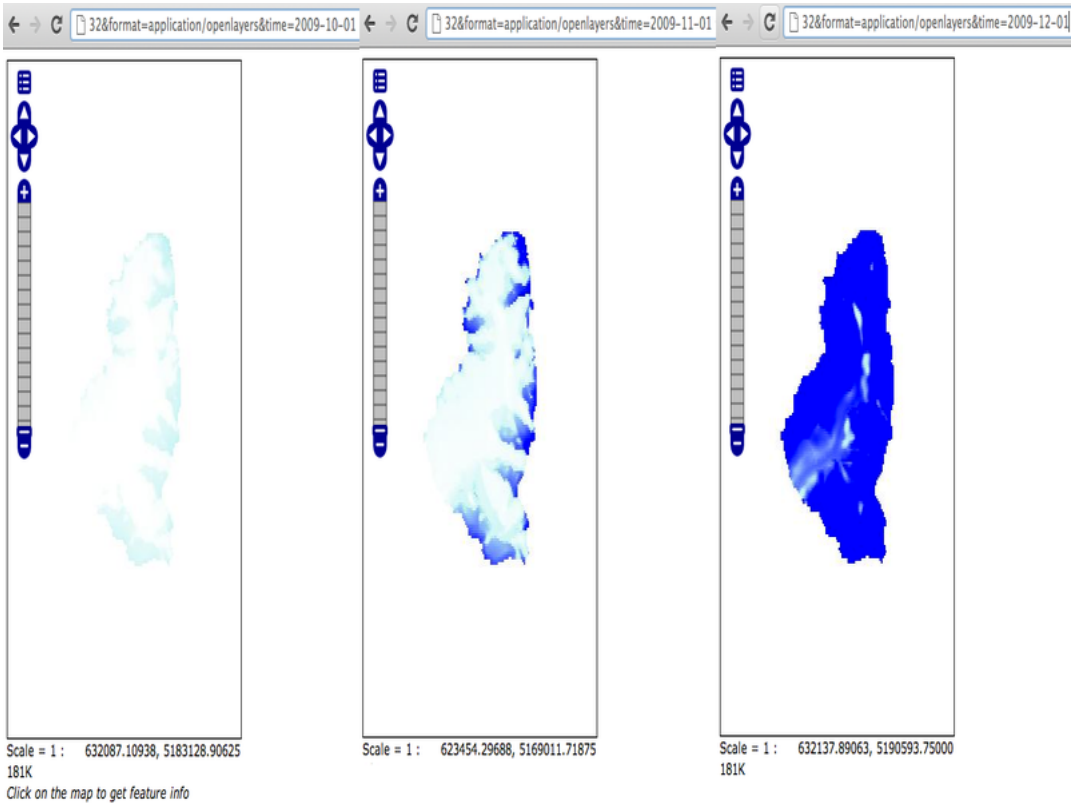
For example if an user wants to obtain the snow coverage images from the months **Oct,Nov,Dec 2009**, pass in each request **&time=2009-10-01**, **&time=2009-11-01** and **&time=2009-12-01**. You can recognize in the three images how the snow coverage changes. Here the color blue means a lot of snow.

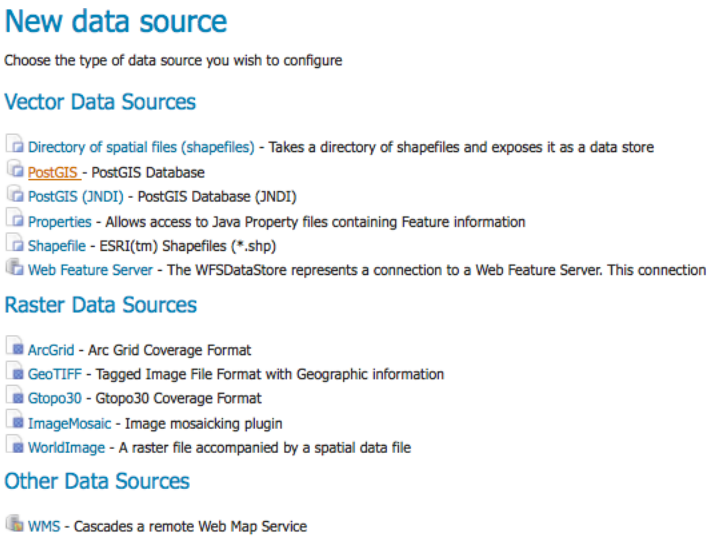
17.9.5 Create and publish a Layer from mosaic indexes:

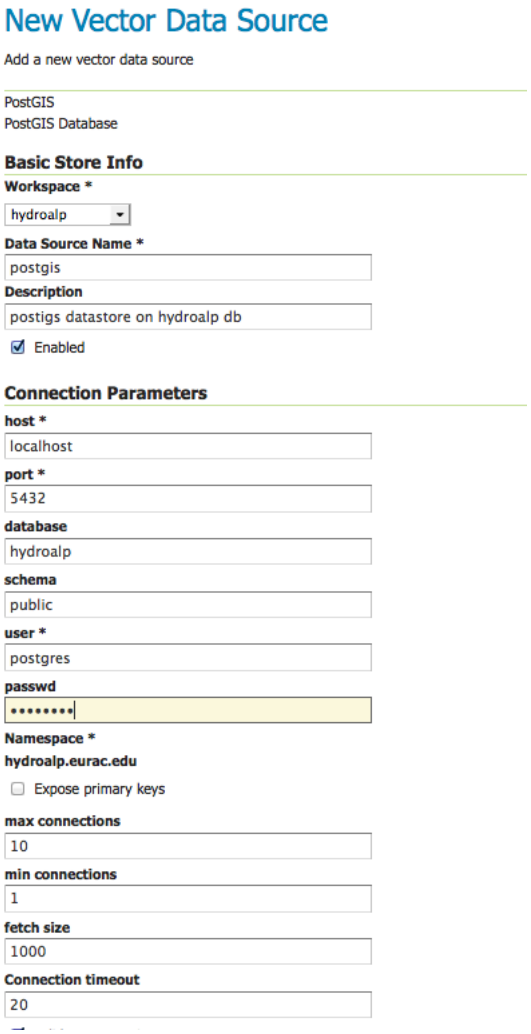
After the previous steps it is also possible to create a layer that represents the spatial indexes of the mosaic. This is an useful feature when handling a large dataset of mosaics with high resolutions granules, since the user can easily get the footprints of the images. In this case will be rendered only the geometries stored on the indexes tables.

Step 1: add a postgis datastore:

and specify the connection parameters

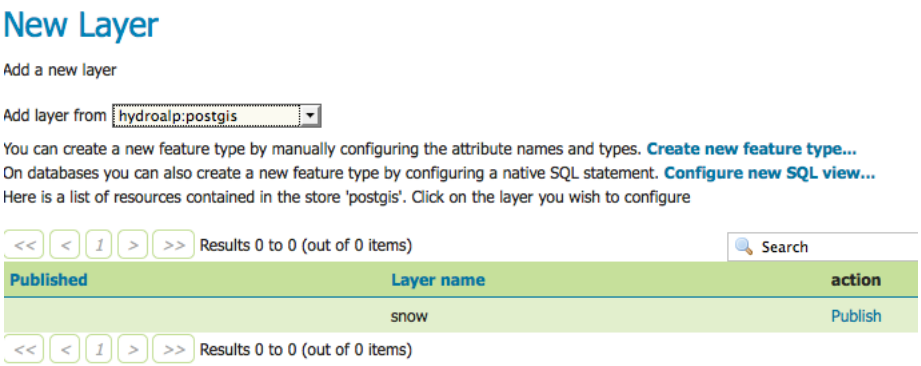

32&format=application/openlayers&time=2009-10-01
32&format=application/openlayers&time=2009-11-01
32&format=application/openlayers&time=2009-12-01
Scale = 1 : 632087.10938, 5183128.90625
181K
Click on the map to get feature info
Scale = 1 : 623454.29688, 5169011.71875
Scale = 1 : 632137.89063, 5190593.75000
181K


New data source
Choose the type of data source you wish to configure
Vector Data Sources
Directory of spatial files (shapefiles) - Takes a directory of shapefiles and exposes it as a data store
PostGIS - PostGIS Database
PostGIS (JNDI) - PostGIS Database (JNDI)
Properties - Allows access to Java Property files containing Feature information
Shapefile - ESRI(tm) Shapefiles (*.shp)
Web Feature Server - The WFSDataStore represents a connection to a Web Feature Server. This connection
Raster Data Sources
ArcGrid - Arc Grid Coverage Format
GeoTIFF - Tagged Image File Format with Geographic information
Gtopo30 - Gtopo30 Coverage Format
ImageMosaic - Image mosaicking plugin
WorldImage - A raster file accompanied by a spatial data file
Other Data Sources
WMS - Cascades a remote Web Map Service


New Vector Data Source
Add a new vector data source
PostGIS
PostGIS Database
Basic Store Info
Workspace *
hydroalp
Data Source Name *
postgis
Description
postigs datastore on hydroalp db
Enabled
Connection Parameters
host *
localhost
port *
5432
database
hydroalp
schema
public
user *
postgres
passwd
Namespace *
hydroalp.eurac.edu
Expose primary keys
max connections
10
min connections
1
fetch size
1000
Connection timeout
20

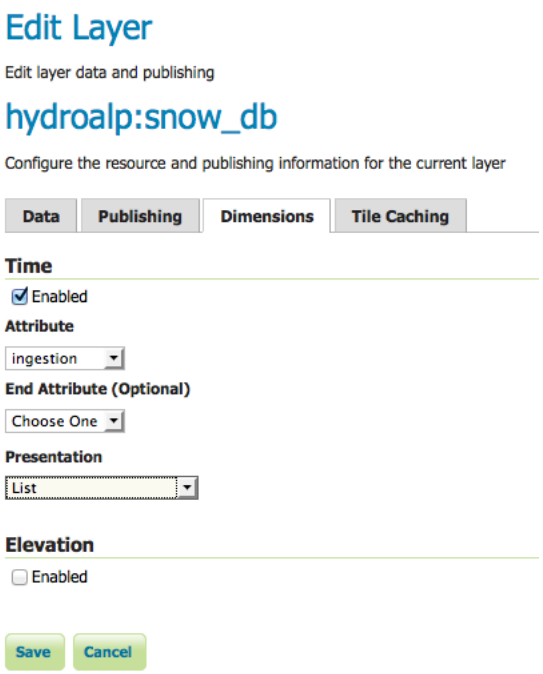
Step 2: add database layer:

Choose from the created datastore the table that you want to publish as a layer.



Step 3: specify dimension:

In the tab Dimension specify the time-variant attribute and the form of presentation.



That's it. Now is possible query this layer too.

17.10 Using the ImageMosaic plugin for raster with time and elevation data

17.10.1 Introduction

This tutorial is the following of *Using the ImageMosaic plugin for raster time-series data* and explains how manage an ImageMosaic using both **Time** and **Elevation** attributes.

The dataset used is a set of raster images used in weather forecast, representing the temperature in a certain zone at different times and elevations.

All the steps explained in chapter *Configurations* of *ImageMosaic* section are still the same.

This tutorial explains just how to configure the **elevationregex.properties** that is an additional configuration file needed, and how to modify the **indexer.properties**.

The dataset used is different so also a fix to the **timeregex.properties** used in the previous tutorial is needed.

Will be shown also how query GeoServer asking for layers specifying both time and elevation dimensions.

The dataset used in the tutorial can be downloaded `Here`

17.10.2 Configuration examples

The additional configurations needed in order to handle also the elevation attributes are:

- Improve the previous version of the *indexer.properties* file
- Add the *elevationregex.properties file* in order to extract the elevation dimension from the filename

indexer.properties:

Here the user can specify the information that needs GeoServer for creating the table in the database.

In this case the time values are stored in the column ingestion as shown in the previous tutorial but now is mandatory specify the elevation column too.

```
Caching=false
TimeAttribute=ingestion
ElevationAttribute=elevation
Schema=*the_geom:Polygon,location:String,ingestion:java.util.Date,elevation:Double
PropertyCollectors=TimestampFileNameExtractorSPI[timeregex](ingestion),
↪DoubleFileNameExtractorSPI[elevationregex](elevation)
```

elevationregex.properties:

Remember that every tif file must follow this naming convention:

```
{coveragename}_{timestamp}_[{elevation}].tif
```

As in the timeregex property file the user must specify the pattern that the elevation in the file name looks like. In this example it consists of 4 digits, a dot '.' and other 3 digits.

an example of filename, that is used in this tutorial is:

```
gfs50kmTemperature20130310T180000000Z_0600.000_.tiff
```

The GeoServer ImageMosaic plugin scans the filename and search for the first occurrence that match with the pattern specified. Here the content of **elevationregex.properties**:

```
regex=(?<=_)(\\d{4}\\.\\d{3})(?=_)
```

timeregex.properties:

As you can see the time in this dataset is specified as ISO8601 format:

```
20130310T180000000Z
```

Instead of the form **yyyymmdd** as in the previous tutorial. So the regex to specify in timeregex.properties is:

```
regex=[0-9]{8}T[0-9]{9}Z(\?!.\*[0-9]{8}T[0-9]{9}Z.\*)
```

17.10.3 Coverage based on filestore

Once the mosaic configuration is ready the store mosaic could be loaded on GeoServer.

The steps needed are the same shown the previous chapter. After the store is loaded and a layer published note the differences in WMS Capabilities document and in the table on postgres.

WMS Capabilities document

The WMS Capabilities document is a bit different, now there is also the dimension **elevation**. In this example both time and elevation dimension are set to **List** .

```
<Dimension name="time" default="current" units="ISO8601">
        2013-03-10T00:00:00.000Z,2013-03-11T00:00:00.000Z,2013-03-12T00:00:00.000Z,
↪2013-03-13T00:00:00.000Z,2013-03-14T00:00:00.000Z,2013-03-15T00:00:00.000Z,2013-03-
↪16T00:00:00.000Z,2013-03-17T00:00:00.000Z,2013-03-18T00:00:00.000Z
</Dimension>
<Dimension name="elevation" default="200.0" units="EPSG:5030" unitSymbol="m">
        200.0,300.0,500.0,600.0,700.0,850.0,925.0,1000.0
</Dimension>
```

The table on postgres

With the elevation support enabled the table on postgres has, for each image, the field **elevation** filled with the elevation value.

Note: The user must create manually the index on the table in order to speed up the search by attribute.

	fid [PK] serial	the_geom geometry	location character varying	ingestion timestamp without time z	elevation double precis
1	1	0103000020E	gfs50kmTemperatu	2013-03-10 18:00:00	200
2	2	0103000020E	gfs50kmTemperatu	2013-03-11 00:00:00	200
3	3	0103000020E	gfs50kmTemperatu	2013-03-11 06:00:00	200
4	4	0103000020E	gfs50kmTemperatu	2013-03-11 12:00:00	200
5	5	0103000020E	gfs50kmTemperatu	2013-03-11 18:00:00	200
6	6	0103000020E	gfs50kmTemperatu	2013-03-12 00:00:00	200
7	7	0103000020E	gfs50kmTemperatu	2013-03-12 06:00:00	200
8	8	0103000020E	gfs50kmTemperatu	2013-03-12 12:00:00	200
9	9	0103000020E	gfs50kmTemperatu	2013-03-12 18:00:00	200
10	10	0103000020E	gfs50kmTemperatu	2013-03-13 00:00:00	200
11	11	0103000020E	gfs50kmTemperatu	2013-03-13 06:00:00	200
12	12	0103000020E	gfs50kmTemperatu	2013-03-13 12:00:00	200
13	13	0103000020E	gfs50kmTemperatu	2013-03-13 18:00:00	200
14	14	0103000020E	gfs50kmTemperatu	2013-03-14 00:00:00	200
15	15	0103000020E	gfs50kmTemperatu	2013-03-14 06:00:00	200
16	16	0103000020E	gfs50kmTemperatu	2013-03-14 12:00:00	200
17	17	0103000020E	gfs50kmTemperatu	2013-03-14 18:00:00	200
18	18	0103000020E	gfs50kmTemperatu	2013-03-15 00:00:00	200
19	19	0103000020E	gfs50kmTemperatu	2013-03-15 06:00:00	200
20	20	0103000020E	gfs50kmTemperatu	2013-03-15 12:00:00	200
21	21	0103000020E	gfs50kmTemperatu	2013-03-15 18:00:00	200

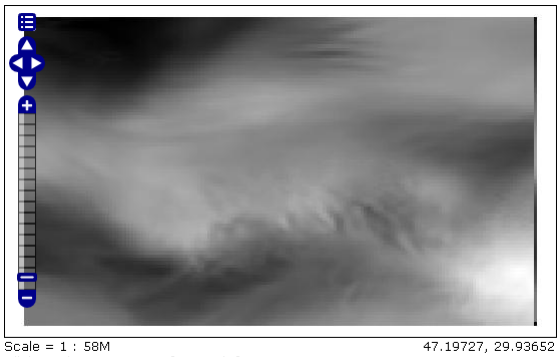
Query layer on timestamp:

In order to display a snapshot of the map at a specific time instant and elevation you have to pass in the request those parameters.

- **&time=** < **pattern** > , as shown before,
- **&elevation=** < **pattern** > where you pass the value of the elevation.

For example if an user wants to obtain the temperature coverage images for the day **2013-03-10 at 6 PM** at elevation **200 meters** must append to the request:

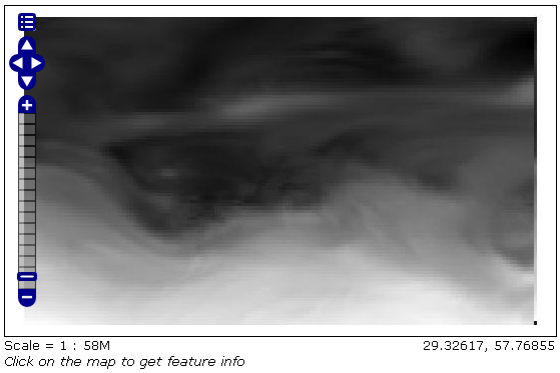
```
&time=2013-03-10T00:00:00.000Z&elevation=200.0
```



Same day at elevation **300.0 meters**:

```
&time=2013-03-10T00:00:00.000Z&elevation=300.0
```

Note that if just the time dimension is append to the request will be displayed the elevation **200 meters** (if present) because of the **default** attribute of the tag `<Dimension name="elevation" ...` in the WMS Capabilities document is set to **200**



17.11 Using the ImageMosaic plugin with footprint mangement

17.11.1 Introduction

This step-by-step tutorial describes how to associate Footprint to a raster data using the ImageMosaic plugin.

Footprint is a Shape used as a Mask for the mosaic. Masking can be useful for hiding pixels which are meaningless, or for enhancing only few regions of the image in respect to others.

This tutorial assumes knowledge of the concepts explained in *ImageMosaic* section.

This tutorial contains two sections:

- The first section, **Configuration**, describes the possible configurations needed to set up an ImageMosaic with footprint.
- The second section, **Examples**, provides examples of configuration of an ImageMosaic with footprint.

17.11.2 Configuration

Footprint can be configured in three different ways:

1. By using for each mosaic granule a *Sidecar File*, a Shapefile with the same name of the granule which contains the footprint for it;
2. By using a single Shapefile called *footprints.shp* which contains all the footprints for each granule; each footprint is associated to a granule with the **location** attribute;
3. By using a file called **footprints.properties** .

The last option should be used when the first two are not available and requires to write the following piece of code inside the **footprints.properties** file:

```
footprint_source=*location of the Shapefile*
footprint_filter=*filter on the Shapefile searching for the attribute associated to␣
↪each granule*
```

For example if a Shapefile called **fakeShapeFile** stores the various footprints in a table like this, where each *Name* attribute is referred to a granule file:

And the associated granules are:

	Name	Shape_Leng	Shape_Area	BUFF_DIST
0	ortho_1-1_1n_s_la087_2010_1	577885.23793900001	4133138857.73999977112	-200.00000000000
1	ortho_2-2_1n_s_la075_2010_1	294998.35735900002	3507709365.03000020981	-200.00000000000
2	ortho_1-1_1n_s_la103_2010_1	315807.69107800000	3903136899.71000003815	-200.00000000000
3	ortho_1-1_1n_s_la071_2010_1	224493.69950700001	1668022813.30999994278	-200.00000000000
4	ortho_1-2_1n_s_la075_2010_1	444899.89763299999	3136445247.32999992371	-200.00000000000
5	ortho_1-1_1n_s_la117_2010_1	235219.58861599999	2778660886.46000003815	-200.00000000000

- ortho_1-1_1n_s_la087_2010_1.tif
- ortho_2-2_1n_s_la075_2010_1.tif
- ortho_1-1_1n_s_la103_2010_1.tif
- and so on ...

The associated **footprints.properties** file must be like this:

```
footprint_source=fakeShapeFile.shp
footprint_filter=Name=strSubstring(granule.location, 0, strLength(granule.location) -␣
↪4)
```

The substring operation is done for comparing the footprint attribute names and the granule names without the *.tif* extension.

There are three possible behaviours for Footprint:

- *None*: simply doesn't use the Footprint and behaves like a standard ImageMosaic layer;
- *Transparent*: adds an alpha band of 0s on the image portions outside of the Footprint making them transparent, typically used for RGB data;
- *Cut*: set the background value on the image portions outside of the Footprint, typically used for GrayScale data.

The behaviour must be set directly on the Layer configuration page.

Another feature of the *Footprint* is the possibility to calculate an **Inset** of the image. *Inset* is a reduction of the footprint border by a value defined by the user which is typically used for removing the compression artifacts. This feature can be achieved by adding the following code inside **footprints.properties** (in case of the first two configurations this file must be added):

```
footprint_inset=*value in the shapefile u.o.m.*
footprint_inset_type=*full/border*
```

Full inset type calculates the inset for each footprint side while **Border** does the same operation but those straight lines that overlap the image bounds are avoided; this last parameter is useful for images already cut in a regular grid.

Each modification of the **footprints.properties** file requires to *Reload* GeoServer. This operation can be achieved by going to *Server Status* and cliking on the *Reload* button on the bottom-right side of the page.

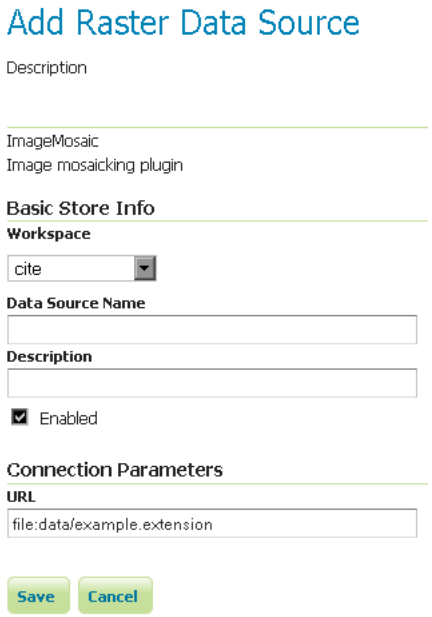
The two datasets used in the tutorial can be downloaded here: Mosaic with a single image which represents Boulder (Colorado), Mosaic with multiple images which represents Italy. The first can be used for testing footprint configuration with a *Sidecar File* and the second for the other two configurations.

17.11.3 Examples

Here are presented a few steps for configuring a new ImageMosaic layer with footprint.

Step 1: Create a new ImageMosaic Layer

As seen before an ImageMosaic data store can be created by going to *Stores* → *Add New Store* → *ImageMosaic*.



An associated Layer can be created by going to *Layers* → *Add New Resource*, choosing the name of the data store created above and then clicking on the *publish* button.

Step 2: Configuring a new Layer for the Mosaic

Inside the new page the only field which is interesting for this tutorial is *FootprintBehavior*:

The user can set one of the three values for the Footprint behaviour as described above.

After that, the user must confirm the modification by clicking on the *Save* button on the bottom side of the page.

Step 3: Example Results

Here are presented the results for each dataset.

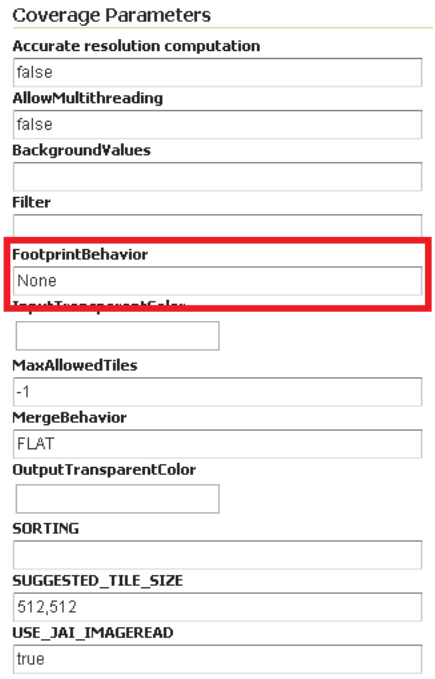
Footprint configured with *Sidecar File*

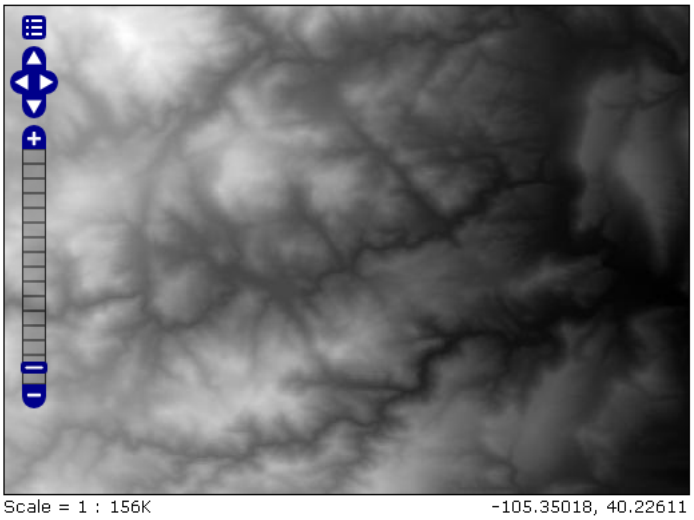
This is an example of mosaic without applying Footprint:

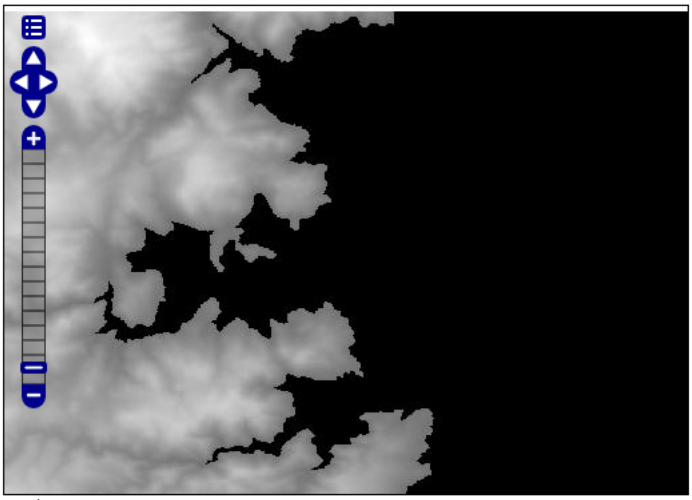
And this is the result of setting **FootprintBehavior** to *Cut*:

Background is gray because in this example the *BackgroundValues* field has been set to -20000.

If an Inset is added, the final mosaic is:


Coverage Parameters
Accurate resolution computation
false
AllowMultithreading
false
Background¥alues
Filter
FootprintBehavior
None
MaxAllowedTiles
-1
MergeBehavior
FLAT
OutputTransparentColor
SORTING
SUGGESTED_TILE_SIZE
512,512
USE_JAI_IMAGEREAD
true


Scale = 1 : 156K
-105.35018, 40.22611


Scale = 1 : 156K
-105.33678, 40.22335


Scale = 1 : 156K
-105.27647, 40.22808

The **footprints.properties** file is:

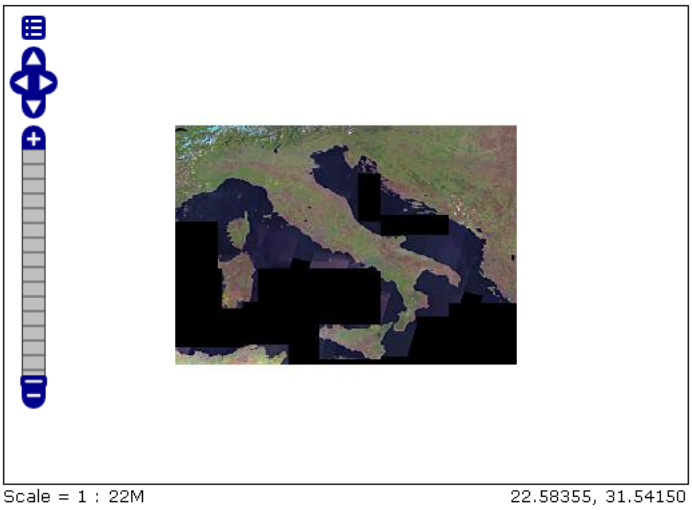
```
footprint_inset=0.01
footprint_inset_type=full
```

Note: Remember that each modification on **footprints.properties** requires a *Reload* of GeoServer for seeing the results.

Note: When configuring this mosaic you must set the *declared CRS* field to "EPSG:4326".

Footprint configured with *footprints.shp*

This is another example of mosaic without Footprint:



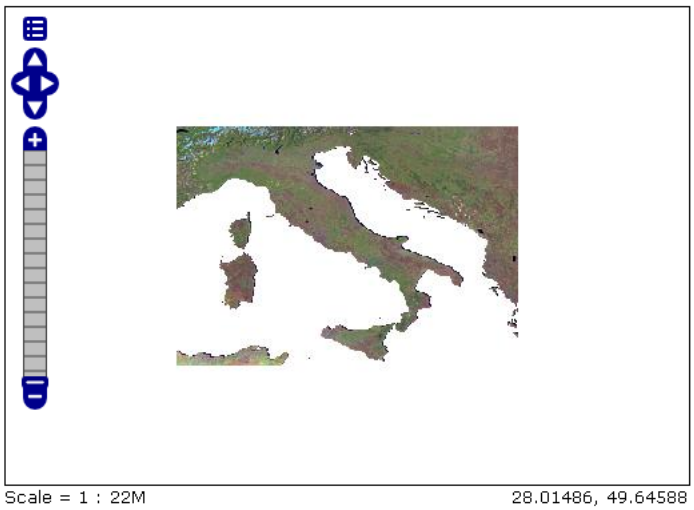
And now after setting **FootprintBehavior** to *Transparent* (no Inset is used) on the Layer:

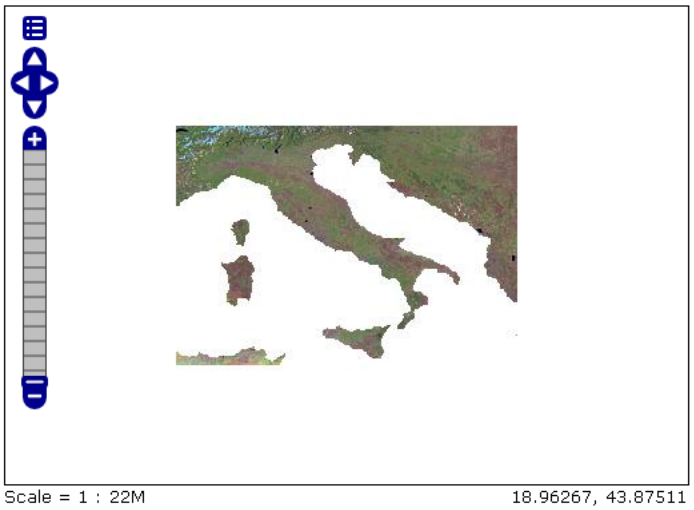
Footprint configured with *footprints.properties*

Note: For testing this functionality the user must rename all the *footprints.xxx* files to *mask.xxx*.

The result of setting **FootprintBehavior** to *Transparent*, Inset type to *border* and Inset value to 0.00001 is:

The **footprints.properties** file is:


Scale = 1 : 22M
28.01486, 49.64588
Click on the map to get feature info


Scale = 1 : 22M
18.96267, 43.87511
Click on the map to get feature info

```
footprint_source=mask.shp
footprint_inset=0.00001
footprint_inset_type=border
```

17.11.4 Raster Masking

From 2.8.x version, GeoServer is able to support also Raster Masks. Those masks can be internal or external (in which case the mask files should use the **.msk** extension), for each file. It is crucial that mask files should have the same pixel size, georeferencing and CRS as the image they are masking.

It must be pointed out that external/internal masks may have overviews like the related original images.

More information about Mask bands may be found at the GDAL Mask Band Page

---

Note: Raster masking is supported for GeoTiff format only and it does not take into account inset definition. The same support is used for ImageMosaic or ImagePyramids made of GeoTiff files.

A **footprints.properties** file that would exploit raster masks would be as follows:

```
footprint_source=raster
```

Note: Raster masks do not support to control inset.

Below you may find an example of configuring a Mosaic with Raster masks:

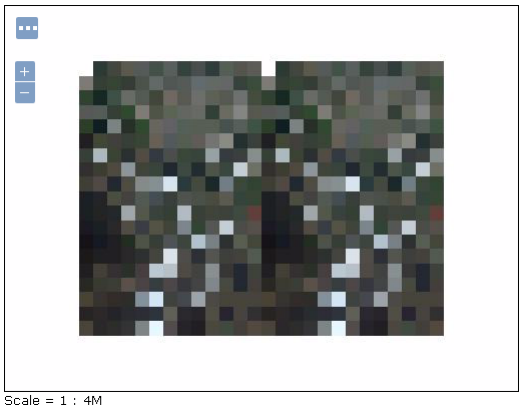
Step 1: Create a new ImageMosaic Layer

Download data from the following `link` and configure an ImageMosaic layer called *rastermask* without changing default configuration parameters.

Zip file contains two images and their related **.msk** files. For this example the two masks are two simple squares.

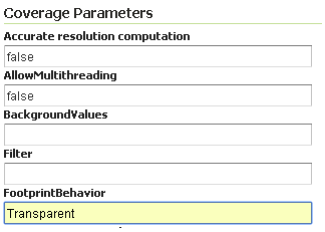
Step 2: Watch the layer using LayerPreview

Go to *LayerPreview* → *rastermask* → *OpenLayers*. The result should be similar to the one below.



Step 3: Change the Footprint Behavior

Change the **FootprintBehavior** parameter to *Transparent*. *Cut* value should not be used since the files are RGB.



Step 4: Check the result

Go to *LayerPreview* → *rastermask* → *OpenLayers*. The result should be changed now.

17.11.5 Multilevel Geometry Masking

From 2.14.x version, GeoServer is able to support also multilevel overviews geometries (A geometry footprint for each overview, being stored on a separate sidecar file).

A **footprints.properties** file that would exploit multiple WKB sidecar files would be as follows:

```
footprint_source=multisidecar
footprintLoaderSPI=org.geotools.coverage.grid.io.footprint.WKBLoaderSPI
overviewsFootprintLoaderSPI=org.geotools.coverage.grid.io.footprint.WKBLoaderSPI
overviewsRoiInRasterSpace=True
overviewsSuffixFormat=_%d
```

Notes:

- *footprintLoaderSPI*: Contains the fully qualified name of the SPI implementation for main footprint loading (Optional property. When not specified, the proper footprint loader will be automatically found by scanning the available SPIs). Currently supported values are:
  - org.geotools.coverage.grid.io.footprint.WKBLoaderSPI for WKB overviews
  - org.geotools.coverage.grid.io.footprint.WKTLoaderSPI for WKT overviews

  - org.geotools.gce.imagemosaic.catalog.ShapeFileLoaderSPI for Shapefile overviews

- *overviewsFootprintLoaderSPI*: Contains the fully qualified name of the SPI implementation for overviews footprints loading (Optional property. When not specified, same loader as footprintLoaderSpi will be used if provided);
- *overviewsRoiInRasterSpace*: Specifies whether the overviews ROI footprint geometrys are in raster space or model space coordinates. (Optional property. Default is False, meaning that overviews footprints are in model space);
- *overviewsSuffixFormat*: Specifies the String format syntax used to define the suffix of the overviews footprints file name. (Optional property. Default is _%d). To give an example, if granule file is R1C1.tif and related 1st overview footprint is stored into R1C1_1.wkt, overviewsSuffixFormat should be _%d. In case 1st overview footprint is stored into R1C1-Ov1.wkt, overviewsSuffixFormat should be -Ov%d.

Same steps of previous section are required to configure an ImageMosaic layer with footprint management.

17.12 Building and using an image pyramid

GeoServer can efficiently deal with large TIFF with overviews, as long as the TIFF is below the 2GB size limit.

Once the image size goes beyond such limit it's time to start considering an image pyramid instead.

An image pyramid builds multiple mosaics of images, each one at a different zoom level, making it so that each tile is stored in a separate file. This comes with a composition overhead to bring back the tiles into a single image, but can speed up image handling as each overview is tiled, and thus a sub-set of it can be accessed efficiently (as opposed to a single GeoTIFF, where the base level can be tiled, but the overviews never are).

This tutorial shows how to build an image pyramid with open source utilities and how to load it into GeoServer. The tutorial assumes you're running at least GeoServer 2.0.2.

17.12.1 Building a pyramid

For this tutorial we have prepared a sample BlueMarble TNG subset in GeoTIFF form. The image is tiled and JPEG compressed, without overviews. Not exactly what you'd want to use for high performance data serving, but good for redistribution and as a starting point to build a pyramid.

In order to build the pyramid we'll use the gdal_retile.py utility, part of the GDAL command line utilities and available for various operating systems (if you're using Microsoft Windows look for FWTools).

The following commands will build a pyramid on disk:

```
mkdir bmpyramid
gdal_retile.py -v -r bilinear -levels 4 -ps 2048 2048 -co "TILED=YES" -co
↪"COMPRESS=JPEG" -targetDir bmpyramid bmreduced.tiff
```

The gdal_retile.py user guide provides a detailed explanation for all the possible parameters, here is a description of the ones used in the command line above:

- *-v*: verbose output, allows the user to see each file creation scroll by, thus knowing progress is being made (a big pyramid construction can take hours)
- *-r bilinear*: use bilinear interpolation when building the lower resolution levels. This is key to get good image quality without asking GeoServer to perform expensive interpolations in memory

- *-levels 4*: the number of levels in the pyramid
- *-ps 2048 2048*: each tile in the pyramid will be a 2048x2048 GeoTIFF
- *-co "TILED=YES"*: each GeoTIFF tile in the pyramid will be inner tiled
- *-co "COMPRESS=JPEG"*: each GeoTIFF tile in the pyramid will be JPEG compressed (trades small size for higher performance, try out it without this parameter too)
- *-targetDir bmpyramid*: build the pyramid in the bmpyramid directory. The target directory must exist and be empty
- *bmreduced.tiff*: the source file

This will produce a number of TIFF files in bmpyramid along with the sub-directories *1*, *2*, *3*, and *4*.

Once that is done, and assuming the GeoServer image pyramid plug-in is already installed, it's possible to create the coverage store by pointing at the directory containing the pyramid and clicking save:

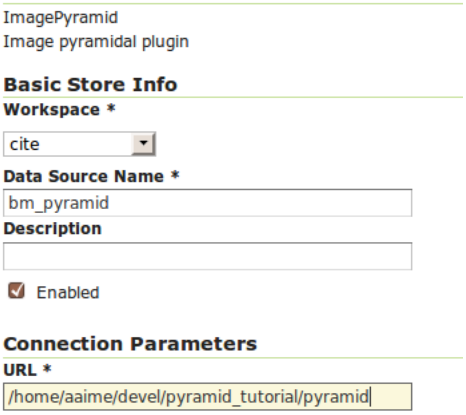


Fig. 17.28: *Configuring a image pyramid store*

When clicking save the store will look into the directory, recognize a *gdal_retile* generated structure and perform some background operations:

- move all tiff files in the root to a newly create directory *0*
- create an image mosaic in all sub-directories (shapefile index plus property file)
- create the root property file describing the whole pyramid structure

Once that is done the user will be asked to choose a coverage, which will be named after the pyramid root directory:

Publish the layer, and then setup the layer parameter *USE_JAI_IMAGEREAD* to *false* to get better scalability:

Submit and go to the preview, the pyramid should be ready to use:

17.12.2 Notes on big pyramids

The code that is auto-creating the pyramid indexes and metadata files might take time to run, especially if:

- the pyramid zero level is composed of many thousands of files
- the system is busy with the disk already and that results in higher times to move all the files to the *0* directory

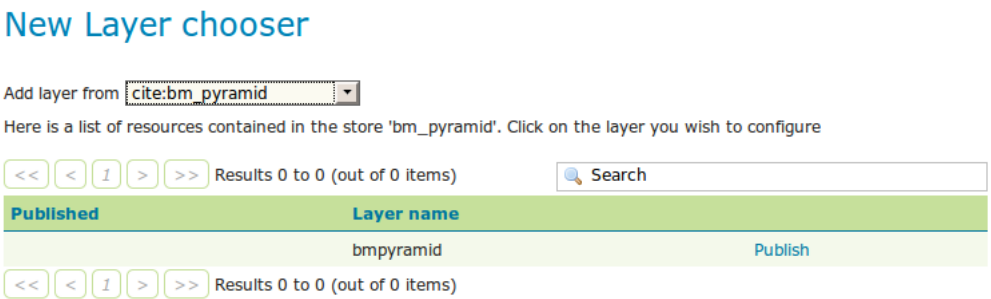


Fig. 17.29: *Choosing the coverage for publishing*

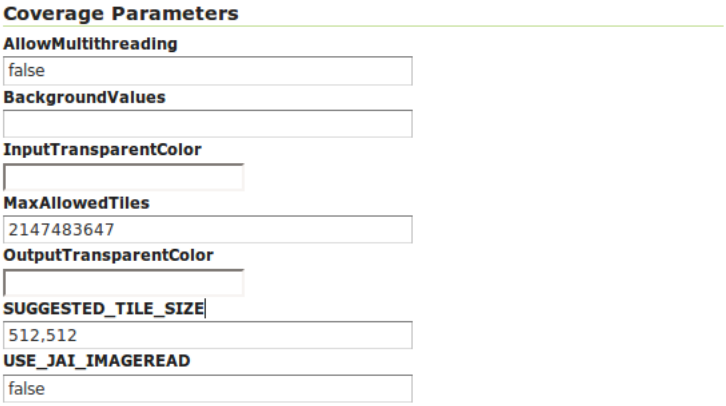


Fig. 17.30: *Tuning the pyramid parameters*

Fig. 17.31: *Previewing the pyramid*

If the delay is too high the request to create the store will time out and might break the pyramid creation. So, in case of very big pyramids consider loosing some of the comfort and creating the *0* directory and moving the files by hand:

```
cd bmpyramid
mkdir 0
mv *.tiff 0
```

17.13 Storing a coverage in a JDBC database

Warning: The screenshots on this tutorial have not yet been updated for the 2.0.x user interface. But most all the rest of the information should be valid, and the user interface is roughly the same, but a bit more easy to use.

17.13.1 Introduction

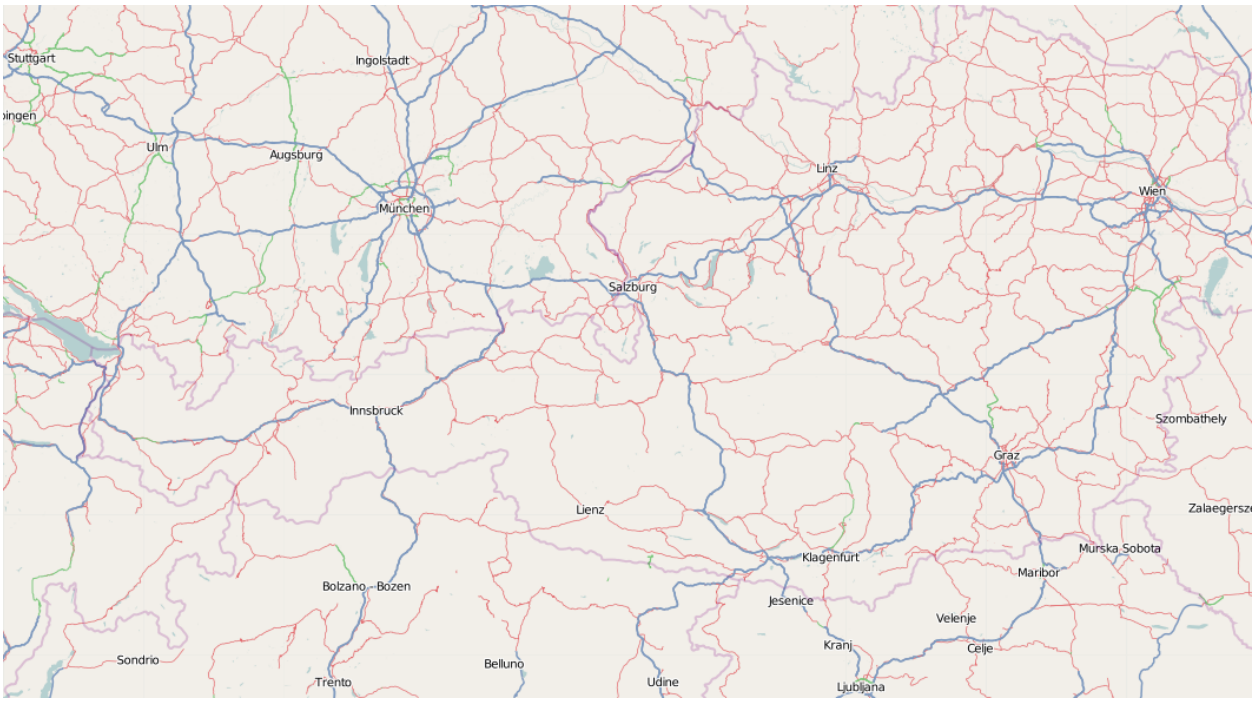
This tutorial describes the process of storing a coverage along with its pyramids in a jdbc database. The ImageMosaic JDBC plugin is authored by Christian Mueller and is part of the geotools library.

The full documentation is available here:http://docs.geotools.org/latest/userguide/library/coverage/jdbc/index.html

This tutorial will show one possible scenario, explaining step by step what to do for using this module in GeoServer (since Version 1.7.2)

17.13.2 Getting Started

We use postgis/postgres as database engine, a database named "gis" and start with an image from openstreetmap. We also need this utility http://www.gdal.org/gdal_retile.html . The best way to install with all dependencies is downloading from here http://fwtools.maptools.org/



Create a working directory, lets call it `working` ,download this image with a right mouse click (Image save as ...) and save it as `start_rgb.png`

Check your image with:

```
gdalinfo start_rgb.png
```

This image has 4 Bands (Red,Green,Blue,Alpha) and needs much memory. As a rule, it is better to use images with a color table. We can transform with **`rgb2pct`** (**`rgb2pct.py`** on Unix).:

```
rgb2pct -of png start_rgb.png start.png
```

Compare the sizes of the 2 files.

Afterwards, create a world file `start.wld` in the `working` directory with the following content.:

```
0.0075471698
0.0000000000
0.0000000000
-0.0051020408
8.9999995849
48.9999999796
```

17.13.3 Preparing the pyramids and the tiles

If you are new to tiles and pyramids, take a quick look here http://northstar-www.dartmouth.edu/doc/idl/html_6.2/Image_Tiling.html

17.13.4 How many pyramids are needed ?

Lets do a simple example. Given an image with 1024x1024 pixels and a tile size with 256x256 pixels.We can calculate in our brain that we need 16 tiles. Each pyramid reduces the number of tiles by a factor of 4. The

first pyramid has 16/4 = 4 tiles, the second pyramid has only 4/4 = 1 tile.

Solution: The second pyramid fits on one tile, we are finished and we need 2 pyramids.

The formula for this:

number of pyramids = log(pixelsize of image) / log(2) - log (pixelsize of tile) / log(2).

Try it: Go to Google and enter as search term "log(1024)/log(2) - log(256)/log(2)" and look at the result.

If your image is 16384 pixels , and your tile size is 512 pixels, it is

log(16384)/log(2) - log(512)/log(2) = 5

If your image is 18000 pixels, the result = 5.13570929. Thake the floor and use 5 pyramids. Remember, the last pyramid reduces 4 tiles to 1 tile, so this pyramid is not important.

If your image is 18000x12000 pixel, use the bigger dimension (18000) for the formula.

For creating pyramids and tiles, use http://www.gdal.org/gdal_retile.html from the gdal project.

The executeable for Windows users is **`gdal_retile.bat`** or only **`gdal_retile`**, Unix users call **`gdal_retile.py`**

Create a subdirectory `tiles` in your `working` directory and execute within the `working` directory:

```
gdal_retile -co "WORLDFILE=YES"  -r bilinear -ps 128 128 -of PNG -levels 2 -targetDir␣
↪tiles start.png
```

What is happening ? We tell gdal_retile to create world files for our tiles (-co "WORLDFILE=YES"), use bilinear interpolation (-r bilinear), the tiles are 128x128 pixels in size (-ps 128 128) , the image format should be PNG (-of PNG), we need 2 pyramid levels (-levels 2) ,the directory for the result is `tiles` (-targetDir tiles) and the source image is `start.png`.

Note: A few words about the tile size. 128x128 pixel is proper for this example. Do not use such small sizes in a production environment. A size of 256x256 will reduce the number of tiles by a factor of 4, 512x512 by a factor of 16 and so on. Producing too much tiles will degrade performance on the database side (large tables) and will also raise cpu usage on the client side (more image operations).

Now you should have the following directories

- `working` containing `start.png` , `start.wld` and a subdirectory `tiles`.
- `working/tiles` containing many `*.png` files and associated `*.wld` files representing the tiles of `start.png`
- `working/tiles/1` containing many `*.png` files and associated `*.wld` files representing the tiles of the first pyramid
- `working/tiles/2` containing many `*.png` files and associated `*.wld` files representing the tiles of the second pyramid

17.13.5 Configuring the new map

The configuration for a map is done in a xml file. This file has 3 main parts.

1. The connect info for the jdbc driver
2. The mapping info for the sql tables
3. Configuration data for the map

Since the jdbc connect info and the sql mapping may be reused by more than one map, the best practice is to create xml fragments for both of them and to use xml entity references to include them into the map xml.

First, find the location of the GEOSERVER_DATA_DIR. This info is contained in the log file when starting GeoServer.:

```
---------------------------------
- GEOSERVER_DATA_DIR: /home/mcr/geoserver-1.7.x/1.7.x/data/release
---------------------------------
```

Put all configuration files into the `coverages` subdirectory of your GeoServer data directory. The location in this example is

`/home/mcr/geoserver-1.7.x/1.7.x/data/release/coverages`

1. Create a file `connect.postgis.xml.inc` with the following content

```
<connect>
  <!-- value DBCP or JNDI -->
  <dstype value="DBCP"/>
  <!--   <jndiReferenceName value=""/>  -->
  <username value="postgres" />
  <password value="postgres" />
  <jdbcUrl value="jdbc:postgresql://localhost:5432/gis" />
  <driverClassName value="org.postgresql.Driver"/>
  <maxActive value="10"/>
  <maxIdle value="0"/>
</connect>
```

The jdbc user is “postgres”, the password is “postgres”, maxActive and maxIdle are parameters of the apache connection pooling, jdbcUrl and driverClassName are postgres specific. The name of the database is “gis”.

If you deploy GeoServer into a J2EE container capable of handling jdbc data sources, a better approach is

```
<connect>
  <!-- value DBCP or JNDI -->
  <dstype value="JNDI"/>
  <jndiReferenceName value="jdbc/mydatasource"/>
</connect>
```

For this tutorial, we do not use data sources provided by a J2EE container.

2. The next xml fragment to create is `mapping.postgis.xml.inc`

```
<!-- possible values: universal,postgis,db2,mysql,oracle -->
<spatialExtension name="postgis"/>
<mapping>
    <masterTable name="mosaic" >
      <coverageNameAttribute name="name"/>
      <maxXAttribute name="maxX"/>
      <maxYAttribute name="maxY"/>
      <minXAttribute name="minX"/>
      <minYAttribute name="minY"/>
      <resXAttribute name="resX"/>
      <resYAttribute name="resY"/>
      <tileTableNameAtribute  name="TileTable" />
      <spatialTableNameAtribute name="SpatialTable" />
    </masterTable>
    <tileTable>
```

```
      <blobAttributeName name="data" />
      <keyAttributeName name="location" />
    </tileTable>
    <spatialTable>
      <keyAttributeName name="location" />
      <geomAttributeName name="geom" />
      <tileMaxXAttribute name="maxX"/>
      <tileMaxYAttribute name="maxY"/>
      <tileMinXAttribute name="minX"/>
      <tileMinYAttribute name="minY"/>
    </spatialTable>
</mapping>
```

The first element `<spatialExtension>` specifies which spatial extension the module should use. "universal" means that there is no spatial db extension at all, meaning the tile grid is not stored as a geometry, using simple double values instead.

This xml fragment describes 3 tables, first we need a master table where information for each pyramid level is saved. Second and third, the attribute mappings for storing image data, envelopes and tile names are specified. To keep this tutorial simple, we will not further discuss these xml elements. After creating the sql tables things will become clear.

3. Create the configuration xml `osm.postgis.xml` for the map (osm for "open street map")

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE ImageMosaicJDBCConfig [
  <!ENTITY mapping PUBLIC "mapping"  "mapping.postgis.xml.inc">
  <!ENTITY connect PUBLIC "connect"  "connect.postgis.xml.inc">]>
<config version="1.0">
  <coverageName name="osm"/>
  <coordsys name="EPSG:4326"/>
  <!-- interpolation 1 = nearest neighbour, 2 = bilinear, 3 = bicubic -->
  <scaleop  interpolation="1"/>
  <verify cardinality="false"/>
  &mapping;
  &connect;
</config>
```

This is the final xml configuration file, including our mapping and connect xml fragment. The coverage name is "osm", CRS is EPSG:4326. `<verify cardinality="false">` means no check if the number of tiles equals the number of rectangles stored in the db. (could be time consuming in case of large tile sets).

This configuration is the hard stuff, now, life becomes easier :-)

17.13.6 Using the java ddl generation utility

The full documentation is here: http://docs.geotools.org/latest/userguide/library/coverage/jdbc/ddl.html

To create the proper sql tables, we can use the java ddl generation utility. This utility is included in the `gt-imagemosaic-jdbc-version.jar`. Assure that this jar file is in your `WEB-INF/lib` directory of your GeoServer installation.

Change to your `working` directory and do a first test:

```
java -jar <your_geoserver_install_dir>/webapps/geoserver/WEB-INF/lib/gt-imagemosaic-
↪jdbc-{version}.jar
```

The reply should be:

```
Missing cmd import | ddl
```

Create a subdirectory `sqlscripts` in your `working` directory. Within the `working` directory, execute:

```
java -jar <your_geoserver_install_dir>/webapps/geoserver/WEB-INF/lib/gt-imagemosaic-
↪jdbc-{version}.jar ddl -config <your geoserver data dir >/coverages/osm.postgis.xml␣
↪-spatialTNPrefix tileosm -pyramids 2 -statementDelim ";" -srs 4326 -targetDir␣
↪sqlscripts
```

Explanation of parameters

parameter	description
ddl	create ddl statements
-config	the file name of our `osm.postgis.xml` file
-pyramids	number of pyramids we want
-statementDelim	The SQL statement delimiter to use
-srs	The db spatial reference identifier when using a spatial extension
-targetDir	output directory for the scripts
-spatialTNPrefix	A prefix for tablenames to be created.

In the directory `working/sqlscripts` you will find the following files after execution:

`createmeta.sql` `dropmeta.sql` `add_osm.sql` `remove_osm.sql`

Note: *IMPORTANT:*

Look into the files `createmeta.sql` and `add_osm.sql` and compare them with the content of `mapping.postgis.xml.inc`. If you understand this relationship, you understand the mapping.

The generated scripts are only templates, it is up to you to modify them for better performance or other reasons. But do not break the relationship to the xml mapping fragment.

17.13.7 Executing the DDL scripts

For user "postgres", databae "gis", execute in the following order:

```
psql -U postgres -d gis  -f createmeta.sql
psql -U postgres -d gis  -f add_osm.sql
```

To clean your database, you can execute `remove_osm.sql` and `dropmeta.sql` after finishing the tutorial.

17.13.8 Importing the image data

The full documentation is here: http://docs.geotools.org/latest/userguide/library/coverage/jdbc/ddl.html

First, the jdbc jar file has to be in the `lib/ext` directory of your java runtime. In my case I had to copy `postgresql-8.1-407.jdbc3.jar`.

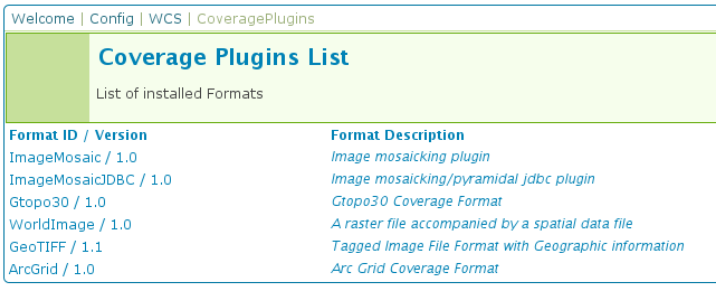
Change to the `working` directory and execute:

```
java -jar <your_geoserver_install_dir>/webapps/geoserver/WEB-INF/lib/gt-imagemosaic-
↪jdbc-{version}.jar import  -config <your geoserver data dir>/coverages/osm.postgis.
↪xml -spatialTNPrefix tileosm -tileTNPrefix tileosm -dir tiles -ext png
```

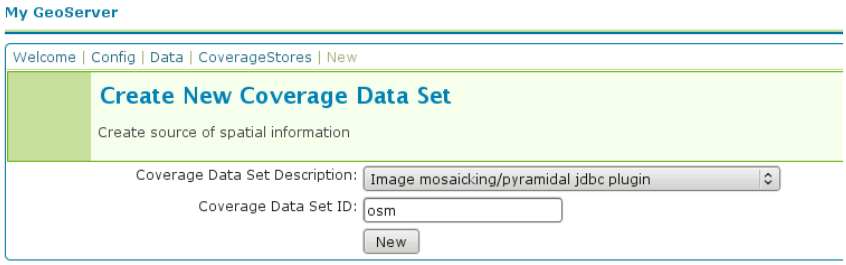
This statement imports your tiles including all pyramids into your database.

17.13.9 Configuring GeoServer

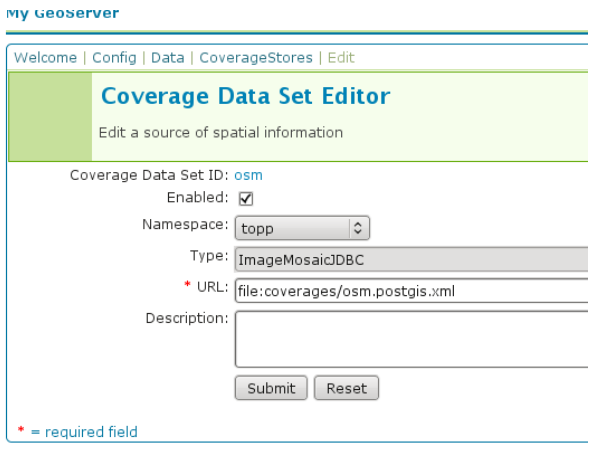
Start GeoServer and log in.Under *Config* → *WCS* → *CoveragePlugins* you should see



If there is no line starting with "ImageMosaicJDBC", the `gt-imagemosiac-jdbc-version.jar` file is not in your `WEB-INF/lib` folder. Go to *Config*→*Data*→*CoverageStores*→*New* and fill in the formular



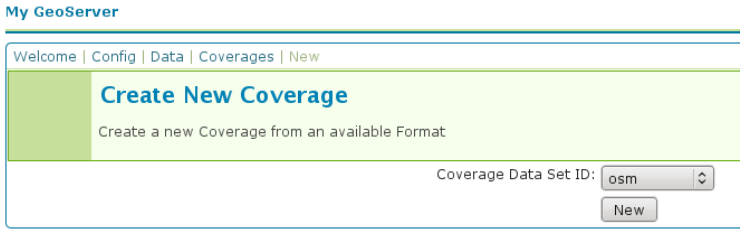
Press *New* and fill in the formular



Press *Submit*.

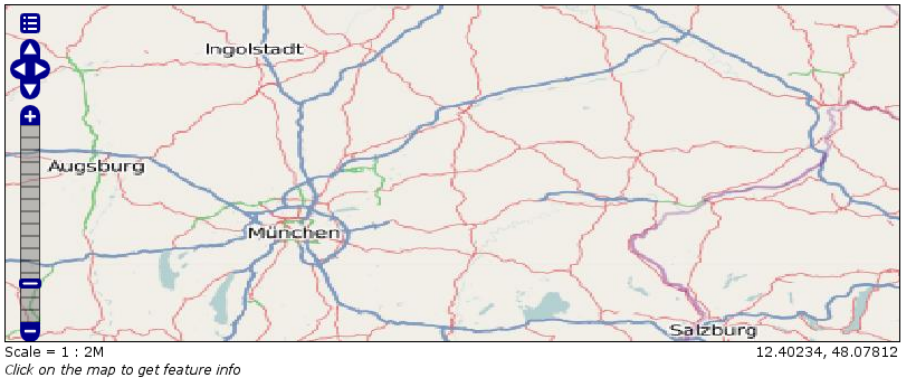
Press *Apply*, then *Save* to save your changes.

Next select *Config→Data→Coverages→New* and select "osm".



Press *New* and you will enter the Coverage Editor. Press *Submit*, *Apply* and *Save*.

Under *Welcome→Demo→Map Preview* you will find a new layer "topp:osm". Select it and see the results



If you think the image is stretched, you are right. The reason is that the original image is georeferenced with EPSG:900913, but there is no support for this CRS in postgis (at the time of this writing). So I used EPSG:4326. For the purpose of this tutorial, this is ok.

17.13.10 Conclusion

There are a lot of other configuration possibilities for specific databases. This tutorial shows a quick cookbook to demonstrate some of the features of this module. Follow the links to the full documentation to dig deeper, especially if you are concerned about performance and database design.

If there is something which is missing, proposals are welcome.

17.14 Using the GeoTools feature-pregeneralized module

Warning: The screenshots on this tutorial have not yet been updated for the 2.0.x user interface. But most all the rest of the information should be valid, and the user interface is roughly the same, but a bit more easy to use.

17.14.1 Introduction

This tutorial shows how to use the geotools feature-pregeneralized module in GeoServer. The feature-pregeneralized module is used to improve performance and lower memory usage and IO traffic.

Note: Vector generalization reduces the number of vertices of a geometry for a given purpose. It makes no sense drawing a polygon with 500000 vertices on a screen. A much smaller number of vertices is enough to draw a topological correct picture of the polygon.

This module needs features with already generalized geometries, selecting the best fit geometry on demand.

The full documentation is available here:http://docs.geotools.org/latest/userguide/library/data/pregeneralized.html

This tutorial will show two possible scenarios, explaining step by step what to do for using this module in GeoServer.

17.14.2 Getting Started

First, find the location of the GEOSERVER_DATA_DIR. This info is contained in the log file when starting GeoServer.:

```
-----------------------------------
- GEOSERVER_DATA_DIR: /home/mcr/geoserver-1.7.x/1.7.x/data/release
-----------------------------------
```

Within this directory, we have to place the shape files. There is already a sub directory `data` which will be used. Within this sub directory, create a directory `streams`.

Within *GEOSERVER_DATA_DIR*`/data/streams` create another sub directory called `0`. (0 meaning "no generalized geometries").

This tutorial is based on on a shape file, which you can download from here `Streams`. Unzip this file into *GEOSERVER_DATA_DIR*`/data/streams/0`.

Look for the `WEB-INF/lib/` directory of your GeoServer installation. There must be a file called `gt-feature-pregeneralized-`*version*`-jar`. This jar file includes a tool for generalizing shape files. Open a cmd line and execute the following:

```
cd <GEOSERVER_DATA_DIR>/data/streams/0
java -jar <GEOSERVER_INSTALLATION>/WEB-INF/lib/gt-feature-pregeneralized-{version}.
↪jar generalize 0/streams.shp . 5,10,20,50
```

You should see the following output:

```
Shape file            0/streams.shp
Target directory      .
Distances             5,10,20,50
% |################################|
```

Now there are four additional directories `5.0` , `10.0` , `20.0` , `50.0` . Look at the size of files with the extension `shp` within these directories, increasing the generalization distance reduces the file size.

Note: The generalized geometries can be stored in additional properties of a feature or the features can be duplicated. Mixed variations are also possible. Since we are working with shape files we have to duplicate

the features.

There are two possibilities how we can deploy our generalized shape files.

1. Deploy hidden (not visible to the user)
2. Deploy each generalized shape file as a separate GeoServer feature

17.14.3 Hidden Deployment

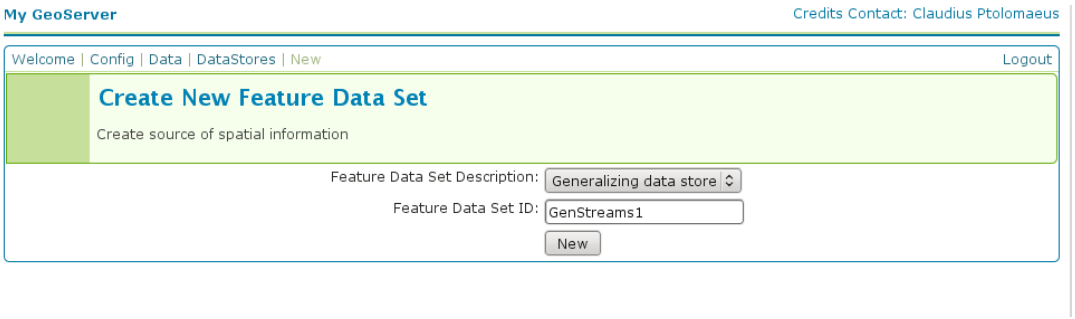
First we need a XML config file

```
<?xml version="1.0" encoding="UTF-8"?>
<GeneralizationInfos version="1.0">
      <GeneralizationInfo dataSourceName="file:data/streams/0/streams.shp"
↪featureName="GenStreams" baseFeatureName="streams" geomPropertyName="the_geom">
            <Generalization dataSourceName="file:data/streams/5.0/streams.shp"
↪distance="5" featureName="streams" geomPropertyName="the_geom"/>
            <Generalization dataSourceName="file:data/streams/10.0/streams.shp"
↪distance="10" featureName="streams" geomPropertyName="the_geom"/>
            <Generalization dataSourceName="file:data/streams/20.0/streams.shp"
↪distance="20" featureName="streams" geomPropertyName="the_geom"/>
            <Generalization dataSourceName="file:data/streams/50.0/streams.shp"
↪distance="50" featureName="streams" geomPropertyName="the_geom"/>
      </GeneralizationInfo>
</GeneralizationInfos>
```

Save this file as `geninfo_shapefile.xml` into *GEOSERVER_DATA_DIR*`/data/streams`.

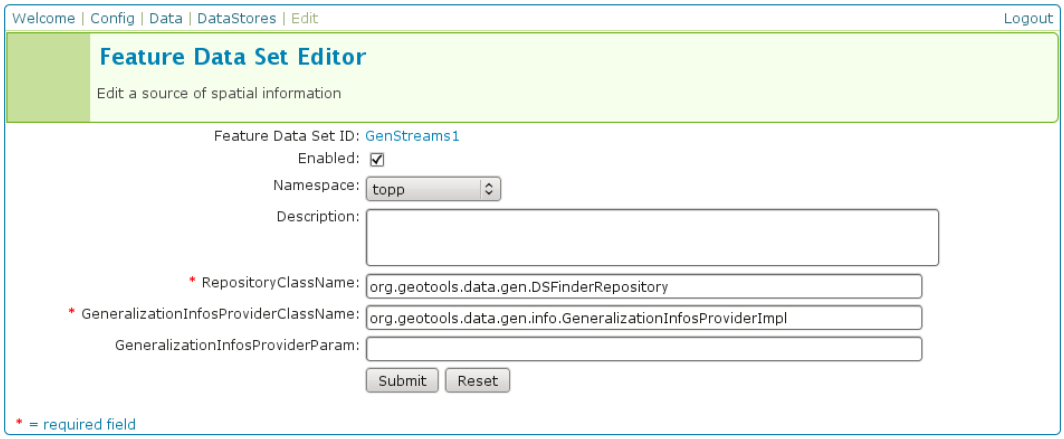
Note: The **dataSourceName** attribute in the XML config is not interpreted as a name, it could be the URL for a shape file or for a property file containing properties for data store creation (e. g. jdbc connect parameters). Remember, this is a hidden deployment and no names are needed. The only *official* name is the value of the attribute **featureName** in the **GeneralizationInfo** Element.

Start GeoServer and go to *Config*→*Data*→*DataStores*→*New* and fill in the form



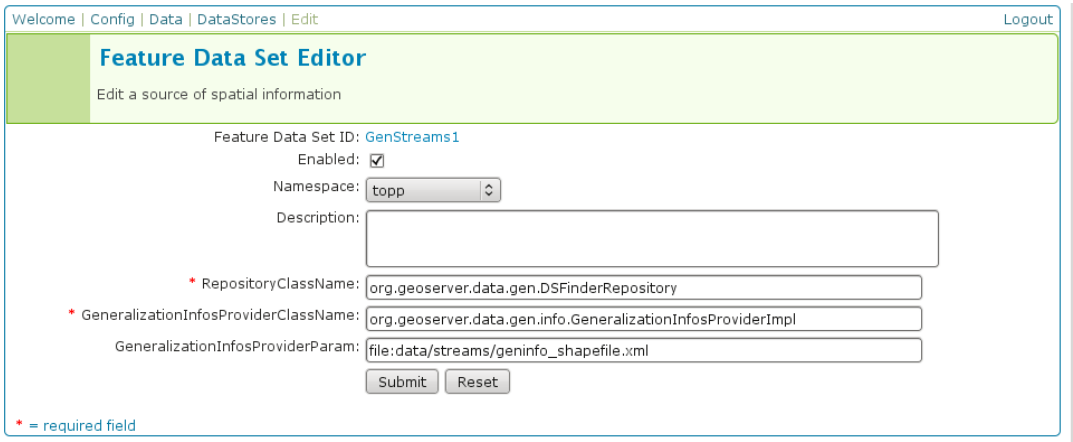
Press *Submit*.

The next form you see is



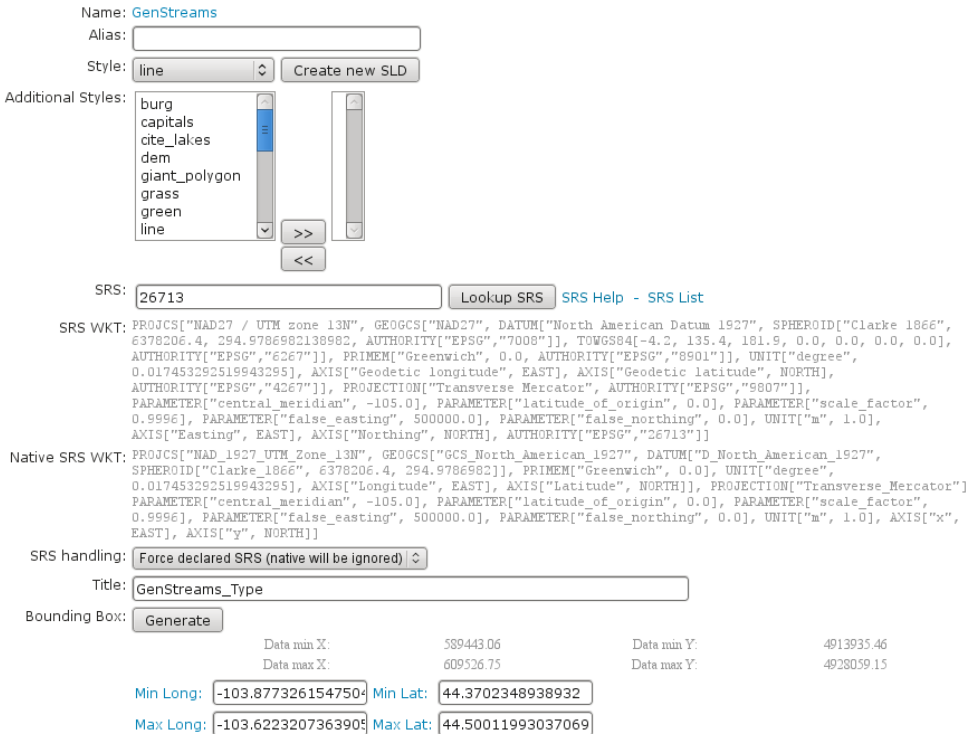
Note: **RepositoryClassName** and **GeneralizationInfosProviderClassName** have default values which suit for GeoTools, not for GeoServer. Change **GeoTools** to **GeoServer** in the package names to instantiate the correct objects for GeoServer. **GeneralizationInfosProviderParam** could be an URL or a datastore from the GeoServer catalog. A datastore is referenced by using *workspacename:datastorename*. This makes sense if you have your own implementation for the **GeneralizationInfosProvider** interface and this implementation reads the infos from a database.

The configuration should look like this



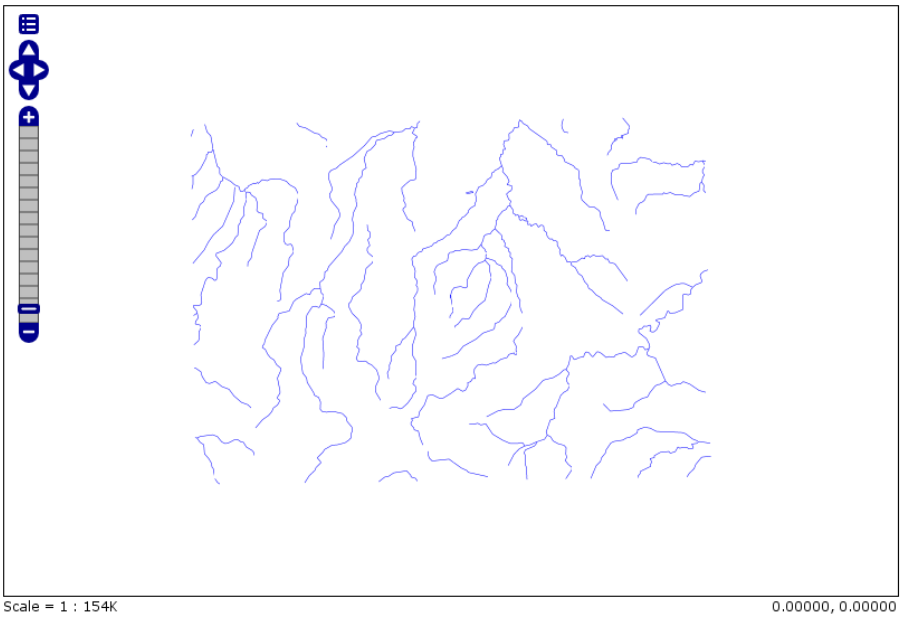
Press *Submit*, afterward a form for the feature type opens.

Alter the **Style** to *line*, **SRS** is *26713* and press the *Generate* button labeled by **Bounding Box**.



Afterward, press *Submit*, *Apply* and *Save*.

Examine the result by pressing "**My GeoServer**, **Demo** and **Map Preview**. In this list there must be an entry **topp:GenStreams**. Press it and you will see



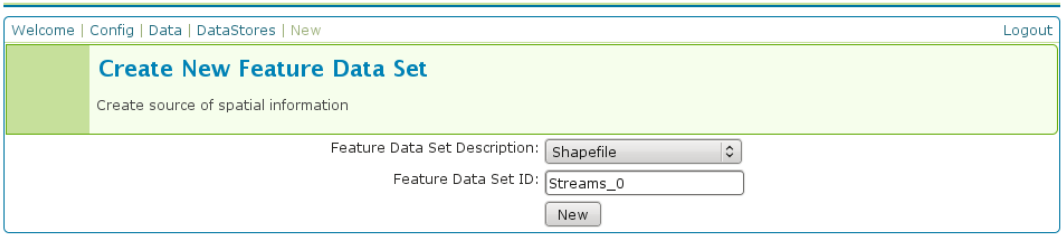
Now start zooming in and out and look at the log file of GeoServer. If the deployment is correct you should see something like this:

```
May 20, 2009 4:53:05 PM org.geotools.data.gen.PreGeneralizedFeatureSource␣
↪logDistanceInfo
INFO: Using generalizsation: file:data/streams/20.0/streams.shp streams the_geom 20.0
```

```
May 20, 2009 4:53:41 PM org.geotools.data.gen.PreGeneralizedFeatureSource␣
↪logDistanceInfo
INFO: Using generalizsation: file:data/streams/5.0/streams.shp streams the_geom 5.0
May 20, 2009 4:54:08 PM org.geotools.data.gen.PreGeneralizedFeatureSource␣
↪logDistanceInfo
INFO: Using generalizsation: file:data/streams/5.0/streams.shp streams the_geom 5.0
May 20, 2009 4:54:09 PM org.geotools.data.gen.PreGeneralizedFeatureSource␣
↪logDistanceInfo
INFO: Using generalizsation: file:data/streams/20.0/streams.shp streams the_geom 20.0
```

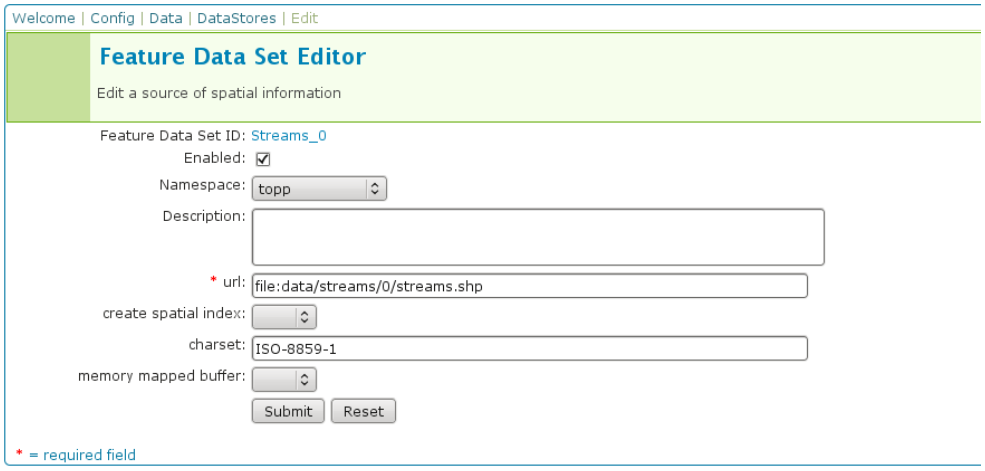
17.14.4 Public Deployment

First we have to configure all our shape files



The **Feature Data Set ID** for the other shape files is

1. Streams_5
2. Streams_10
3. Streams_20
4. Streams_50



The **URL** needed for the other shape files

1. `file:data/streams/5.0/streams.shp`
2. `file:data/streams/10.0/streams.shp`
3. `file:data/streams/20.0/streams.shp`

4. `file:data/streams/50.0/streams.shp`

Each feature needs an **Alias**, here it is *streams_0*. For the other shape files use

1. streams_5
2. streams_10
3. streams_20
4. streams_50

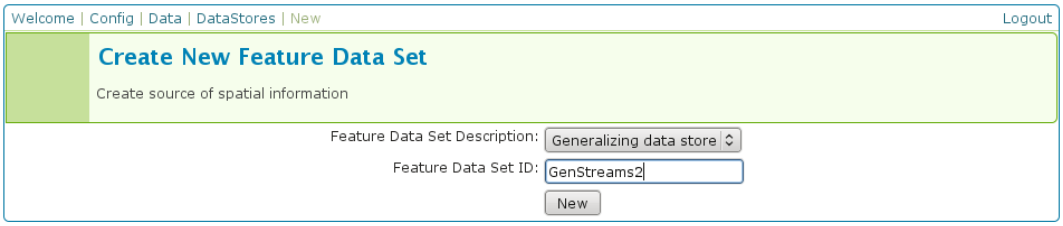
Check the result by pressing *My GeoServer*, *Demo* and *Map Preview*. You should see your additional layers.

No we need another XML configuration file

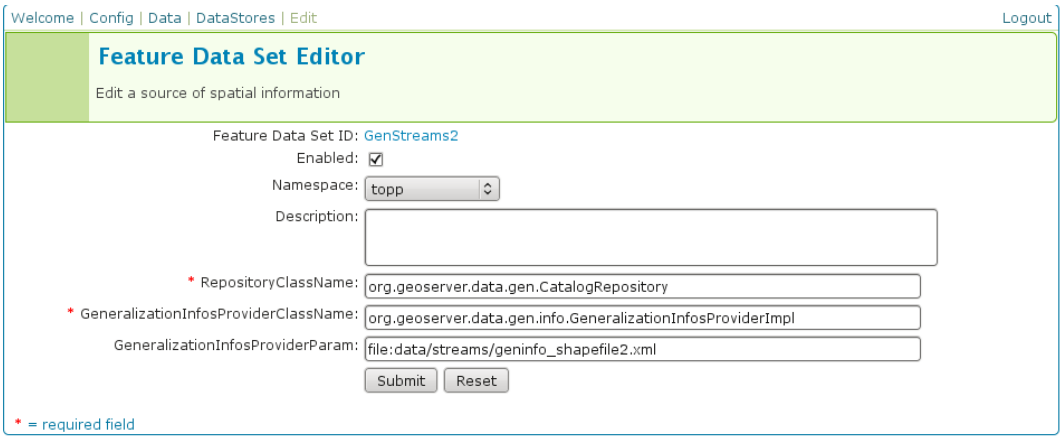
```
<?xml version="1.0" encoding="UTF-8"?>
<GeneralizationInfos version="1.0">
      <GeneralizationInfo dataSourceNameSpace="topp" dataSourceName="Streams_0"
↪featureName="GenStreams2" baseFeatureName="streams" geomPropertyName="the_geom">
            <Generalization dataSourceNameSpace="topp" dataSourceName="Streams_5"
↪distance="5" featureName="streams" geomPropertyName="the_geom"/>
            <Generalization dataSourceNameSpace="topp" dataSourceName="Streams_10"
↪distance="10" featureName="streams" geomPropertyName="the_geom"/>
            <Generalization dataSourceNameSpace="topp" dataSourceName="Streams_20"
↪distance="20" featureName="streams" geomPropertyName="the_geom"/>
            <Generalization dataSourceNameSpace="topp" dataSourceName="Streams_50"
↪distance="50" featureName="streams" geomPropertyName="the_geom"/>
      </GeneralizationInfo>
</GeneralizationInfos>
```

Save this file as `geninfo_shapefile2.xml` into *GEOSERVER_DATA_DIR*`/data/streams`.

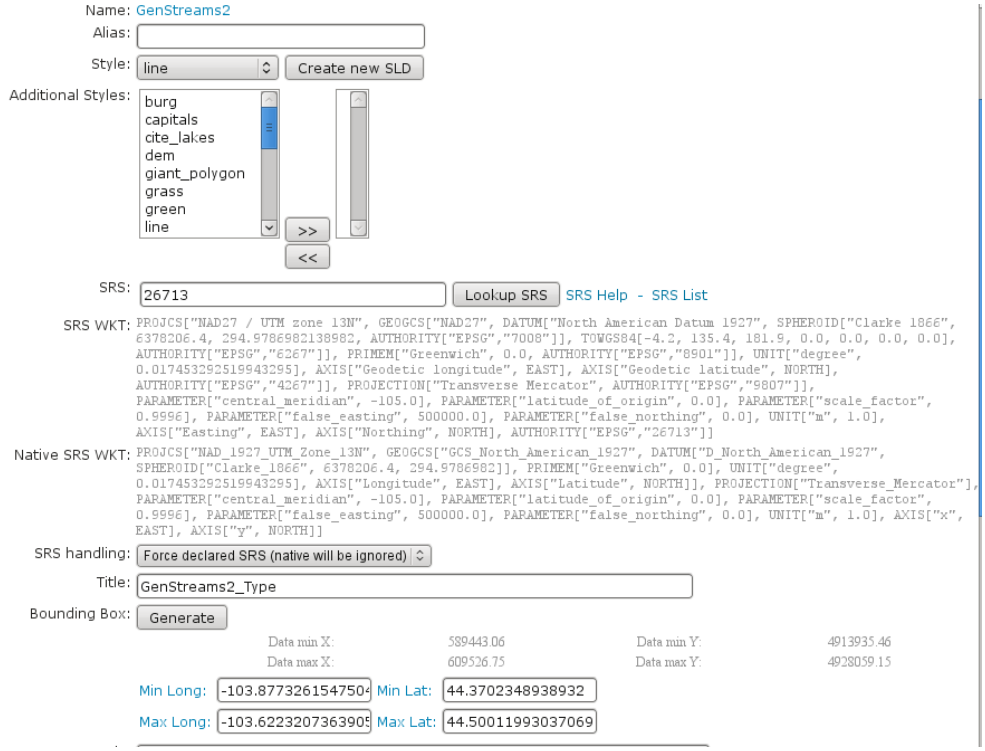
Create the pregeneralized datastore



Now we use the **CatalogRepository** class to find our needed data stores



Last step



In the *Map Preview* you should find **topp:GenStreams2** and all other generalizations. Test in the same manner we discussed in the hidden deployment and you should see something like this in the GeoServer

log:

```
May 20, 2009 6:11:06 PM org.geotools.data.gen.PreGeneralizedFeatureSource␣
↪logDistanceInfo
INFO: Using generalizsation: Streams_20 streams the_geom 20.0
May 20, 2009 6:11:08 PM org.geotools.data.gen.PreGeneralizedFeatureSource␣
↪logDistanceInfo
INFO: Using generalizsation: Streams_10 streams the_geom 10.0
May 20, 2009 6:11:12 PM org.geotools.data.gen.PreGeneralizedFeatureSource␣
↪logDistanceInfo
INFO: Using generalizsation: Streams_10 streams the_geom 10.0
```

17.14.5 Conclusion

This is only a very simple example using shape files. The plugin architecture allows you to get your data and generalizations from anywhere. The used dataset is a very small one, so you will not feel a big difference in response time. Having big geometries (in the sense of many vertices) and creating maps with some different layers will show the difference.

17.15 Setting up a JNDI connection pool with Tomcat

This tutorial walks the reader through the procedures necessary to setup a Oracle JNDI connection pool in Tomcat 6 and how to retrieve it from GeoServer. In the last section other two examples of configuration are described with PostGIS and SQLServer.

17.15.1 Tomcat setup

In order to setup a connection pool Tomcat needs a JDBC driver and the necessary pool configurations.

First off, you need to find the JDBC driver for your database. Most often it is distributed on the web site of your DBMS provider, or available in the installed version of your database. For example, a Oracle XE install on a Linux system provides the driver at `/usr/lib/oracle/xe/app/oracle/product/10.2.0/server/jdbc/lib/ojdbc14.jar`, and that file needs to be moved into Tomcat shared libs directory, *`TOMCAT_HOME`*`/lib`

Note: be careful to remove the jdbc driver from the GeoServer WEB-INF/lib folder when copied to the Tomcat shared libs, to avoid issues in JNDI DataStores usage.

Once that is done, the Tomcat configuration file *`TOMCAT_HOME`*`/conf/context.xml` needs to be edited in order to setup the connection pool. In the case of a local Oracle XE the setup might look like:

```
<Context>
   ...
   <Resource name="jdbc/oralocal"
      auth="Container"
      type="javax.sql.DataSource"
      driverClassName="oracle.jdbc.driver.OracleDriver"
      url="jdbc:oracle:thin:@localhost:1521:xe"
      username="xxxxx" password="xxxxxx"
      maxActive="20"
      initialSize="0"
```

```
      minIdle="0"
      maxIdle="8"
      maxWait="10000"
      timeBetweenEvictionRunsMillis="30000"
      minEvictableIdleTimeMillis="60000"
      testWhileIdle="true"
      poolPreparedStatements="true"
      maxOpenPreparedStatements="100"
      validationQuery="SELECT SYSDATE FROM DUAL"
      maxAge="600000"
      rollbackOnReturn="true"
      />
</Context>
```

The example sets up a connection pool connecting to the local Oracle XE instance. The pool configuration shows is quite full fledged:

- at most 20 active connections (max number of connection that will ever be used in parallel)
- at most 3 connections kept in the pool unused
- prepared statement pooling (very important for good performance)
- at most 100 prepared statements in the pool
- a validation query that double checks the connection is still alive before actually using it (this is not necessary if there is guarantee the connections will never drop, either due to the server forcefully closing them, or to network/maintenance issues).

Warning: Modify following settings only if you really know what you are doing. Using too low values for `removedAbandonedTimeout` and `minEvictableIdleTimeMillis` may result in connection failures, if so try to setup `logAbandoned` to `true` and check your `catalina.out` log file.

Other parameters to setup connection pool:

- timeBetweenEvictionRunsMillis (default -1) The number of milliseconds to sleep between runs of the idle object evictor thread. When non-positive, no idle object evictor thread will be run.
- numTestsPerEvictionRun (default 3) The number of objects to examine during each run of the idle object evictor thread (if any).
- minEvictableIdleTimeMillis (default 1000 * 60 * 30) The minimum amount of time an object may sit idle in the pool before it is eligable for eviction by the idle object evictor (if any).
- removeAbandoned (default false) Flag to remove abandoned connections if they exceed the removeAbandonedTimout. If set to true a connection is considered abandoned and eligible for removal if it has been idle longer than the removeAbandonedTimeout. Setting this to true can recover db connections from poorly written applications which fail to close a connection.
- removeAbandonedTimeout (default 300) Timeout in seconds before an abandoned connection can be removed.
- logAbandoned (default false) Flag to log stack traces for application code which abandoned a Statement or Connection.

For more information about the possible parameters and their values refer to the DBCP documentation.

17.15.2 GeoServer setup

Login into the GeoServer web administration interface and configure the datastore.

First, choose the *Oracle (JNDI)* datastore and give it a name:

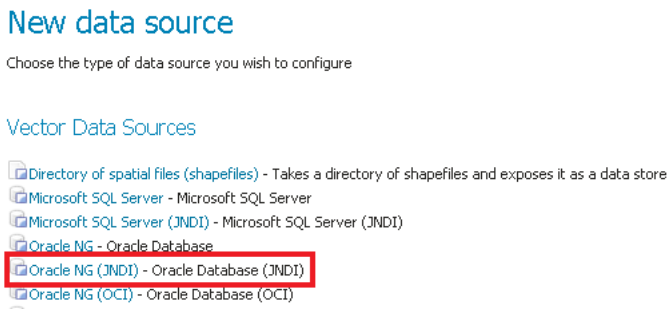


Fig. 17.32: *Choosing a JNDI enabled datastore*

Then, configure the connection parameters so that the JNDI path matches the one specified in the Tomcat configuration:

When you are doing this, make sure the *schema* is properly setup, or the datastore will list all the tables it can find in the schema it can access. In the case of Oracle the schema is usually the user name, upper cased.

Once the datastore is accepted the GeoServer usage proceeds as normal.

17.15.3 Other examples

Configuring a PostgreSQL connection pool

In this example a PostgreSQL connection pool will be configured.

For configuring the JNDI pool you need to remove the Postgres JDBC driver (it should be named `postgresql-X.X-XXX.jdbc3.jar`) from the GeoServer `WEB-INF/lib` folder and put it into the *`TOMCAT_HOME`*`/lib` folder.

Then the following code must be written in the Tomcat configuration file *`TOMCAT_HOME`*`/conf/context.xml`

```
<Context>
        <Resource name="jdbc/postgres"
      auth="Container"
      type="javax.sql.DataSource"
      driverClassName="org.postgresql.Driver"
      url="jdbc:postgresql://localhost:5432/test"
      username="xxxxx" password="xxxxxx"
      maxActive="20"
      initialSize="0"
      minIdle="0"
      maxIdle="8"
      maxWait="10000"
      timeBetweenEvictionRunsMillis="30000"
      minEvictableIdleTimeMillis="60000"
      testWhileIdle="true"
      validationQuery="SELECT 1"
      maxAge="600000"
```

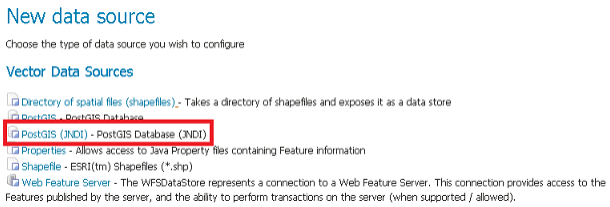
Fig. 17.33: *Configuring the JNDI connection*

```
      rollbackOnReturn="true"
    />
</Context>
```

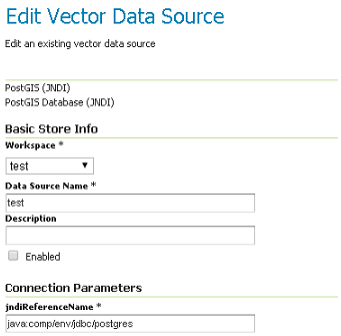
GeoServer setup

Login into the GeoServer web administration interface.

First, choose the *PostGIS (JNDI)* datastore and give it a name:



Then configure the associated params:



Configuring a SQLServer connection pool

For configuring the connection pool for SQLServer you need to configure the SQLServer drivers as explained in the *Microsoft SQL Server* section and put the jar file into the *TOMCAT_HOME*/lib folder.

Then the following code must be written in the Tomcat configuration file *TOMCAT_HOME*/conf/context.xml

```
<Context>
   ...
      <Resource name="jdbc/sqlserver"
      auth="Container"
      type="javax.sql.DataSource"
      driverClassName="com.microsoft.sqlserver.jdbc.SQLServerDriver"
      url="jdbc:sqlserver://localhost:1433;databaseName=test;user=admin;
↪password=admin;"
      username="admin" password="admin"
      maxActive="20"
      initialSize="0"
      minIdle="0"
      maxIdle="8"
```

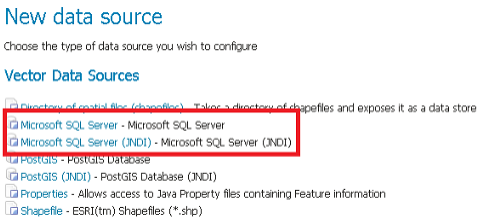
```
      maxWait="10000"
      timeBetweenEvictionRunsMillis="30000"
      minEvictableIdleTimeMillis="60000"
      testWhileIdle="true"
      poolPreparedStatements="true"
      maxOpenPreparedStatements="100"
      validationQuery="SELECT SYSDATE FROM DUAL"
      maxAge="600000"
      rollbackOnReturn="true"
      />
</Context>
```

Note: Note that database name, username and password must be defined directly in the URL.

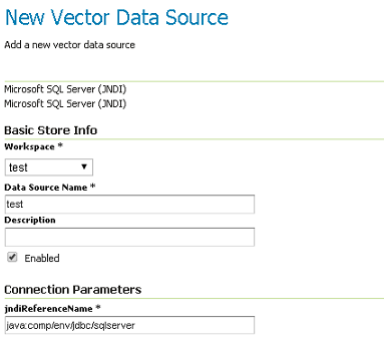
GeoServer setup

Login into the GeoServer web administration interface.

First, choose the *Microsoft SQL Server (JNDI)* datastore and give it a name:



Then configure the associated params:



17.16 geoserver on JBoss

This tutorial documents how to install various versions of geoserver onto various versions of JBoss.

17.16.1 geoserver 2.7.0 on JBoss AS 5.1

To install geoserver onto JBoss AS 5.1, the following is required:

1. Create the file `jboss-classloading.xml` with the following content then copy it into the `WEB-INF` directory in the geoserver.war:

```
<classloading xmlns="urn:jboss:classloading:1.0"
        name="geoserver.war"
        domain="GeoServerDomain">
</classloading>
```

2. Extract the `hsqldb-2.2.8.jar` file from the `WEB-INF/lib` directory from the geoserver.war and copy it as `hsqldb.jar` to the `common/lib` directory in the JBoss deployment.
3. Add the following text to the `WEB-INF/web.xml` file in the geoserver.war so that JBoss logging does not end up in the geoserver.log:

```
<context-param>
        <param-name>RELINQUISH_LOG4J_CONTROL</param-name>
        <param-value>true</param-value>
</context-param>
```

Python Module Index

j