



Services Standard Build User Guide

SSB v6, July 2018

This document is intended to be a user-friendly guide explaining how to set-up the Services Standard Build, as well as its applications.

Document Revision History

Revision Date	Written/Edited By	Comments
February 2013	Blake Bowen	Initial Creation (Current IdentityIQ version: 6.0)
March 2013	Tina Timmerman	1 st Revision
April 2013	Brendon Jones	2 nd Revision
June 2013	Blake Bowen	Final Revision for SSB 1.2, posted to Compass
August 2013	Blake Bowen	Updated Compass links to new Compass
January 2015	Blake Bowen	Updated with version 1.5 features, new Compass links, and further clarification.
June 2016	Blake Bowen	Updated Compass link to point to SSD get started page
September 2016	Paul Wheeler	Updated with new features added in v2.
October 2016	Paul Wheeler	Minor corrections for revision 2.0.1.
December 2016	Paul Wheeler	Example custom script file names modified so that they do not execute unless renamed.
January 2017	Paul Wheeler	Support for environment-specific build.properties files clarified.
February 2017	Paul Wheeler	Minor restructure and updates for SSB v3 release
June 2017	Justin Chophonis, Paul Wheeler	Formatting and structure changes. Updates for SSB v4 release
December 2017	Paul Wheeler	Added plugin build information for SSB v5, added Deprecation Scanner in the Build Checks.
July 2018	Paul Wheeler	Added functionality for subset builds, IdentityIQ keystore file deployment and "secret" target.properties files for SSB v6. Expansion of efixes in timestamp order. Additional Build Check for Workflow trace setting. Added information on the Dependency Check utility.

© Copyright 2018 SailPoint Technologies, Inc., All Rights Reserved.

SailPoint Technologies, Inc. makes no warranty of any kind with regard to this manual, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. SailPoint Technologies shall not be liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Restricted Rights Legend. All rights are reserved. No part of this document may be photocopied, reproduced, or translated to another language without the prior written consent of SailPoint Technologies. The information contained in this document is subject to change without notice.

Use, duplication or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c) (1) (ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 for DOD agencies, and subparagraphs (c) (1) and (c) (2) of the Commercial Computer Software Restricted Rights clause at FAR 52.227-19 for other agencies.

Regulatory/Export Compliance. The export and reexport of this software is controlled for export purposes by the U.S. Government. By accepting this software and/or documentation, licensee agrees to comply with all U.S. and foreign export laws and regulations as they relate to software and related documentation. Licensee will not export or reexport outside the United States software or documentation, whether directly or indirectly, to any Prohibited Party and will not cause, approve or otherwise intentionally facilitate others in so doing. A Prohibited Party includes: a party in a U.S. embargoed country or country the United States has named as a supporter of international terrorism; a party involved in proliferation; a party identified by the U.S. Government as a Denied Party; a party named on the U.S. Government's Entities List; a party prohibited from participation in export or reexport transactions by a U.S. Government General Order; a party listed by the U.S. Government's Office of Foreign Assets Control as ineligible to participate in transactions subject to U.S. jurisdiction; or any party that licensee knows or has reason to know has violated or plans to violate U.S. or foreign export laws or regulations. Licensee shall ensure that each of its software users complies with U.S. and foreign export laws and regulations as they relate to software and related documentation.

Trademark Notices. Copyright © 2018 SailPoint Technologies, Inc. All rights reserved. SailPoint, the SailPoint logo, SailPoint IdentityIQ, and SailPoint Identity Analyzer are trademarks of SailPoint Technologies, Inc. and may not be used without the prior express written permission of SailPoint Technologies, Inc. All other trademarks shown herein are owned by the respective companies or persons indicated.

Table of Contents

Services Standard Build Overview	7
SSB Components	7
Apache Ant	7
Ant Contrib	8
Catalina-Ant	8
Custom Ant Tasks	8
Process Overview	9
Downloading the Services Standard Build	10
Folder Structure	10
.keep Files	13
Exporting Custom Objects	13
Running the Export Script	14
Build Structure Set-up	17
Configuration Objects	17
IdentityIQ Product Files	17
Plugins	17
Build/Compilation of Plugins	17
Automatic Deployment of Plugins	18
JDBC Drivers	19
Build Configuration	20
Configuring the build.properties file	20
Supporting multiple platforms (Windows/Linux/Unix) for different environments	25
Non-default spadmin password and importing artifacts	26
Setting up environment-specific properties files	26
Configuring iiq.properties files	26
Configuring target.properties files	28
Configuring “secret” target.properties files for storing sensitive token values	29
Configuring Subset Builds with includefiles.properties files	30
Configuring ignorefiles.properties files	31
Configuring log4j.properties files	31
Configuring deployment of encryption keys for each environment	32
Setting the environment name for a build	33

Using the SPTARGET environment variable to specify the build environment	33
Setting the target variables by editing servers.properties	33
Executing the Build	34
Executing a Repeatable, Initial Build of IdentityIQ with SSB	35
Initial Build Prerequisites	35
Initial Build Target Chaining.....	37
Installation	37
Removal	37
Dev targets explained.....	38
<i>No target</i> (just entering “build” into a windows terminal or “./build.sh” into a Linux terminal).....	38
main.....	38
clean.....	38
cleanWeb	38
createdb	39
cycle	39
dropdb	39
dist.....	40
dependency-check.....	40
deploy.....	40
document.....	40
down.....	40
extenddb.....	41
export	41
import-all.....	41
import-custom.....	41
import-lcm.....	42
import-stock	42
import (deprecated)	42
importcycle	42
importdynamic	43
importjava	43
initial-build.....	43
patchdb.....	43

runSql.....	43
runUpgrade.....	44
up	45
war	45
Build Checks	45
Controlling Build Checks.....	45
Available Build Checks	45
Checks after SSB Build Expansion Phase.....	45
Checks after SSB Token Substitution Phase.....	46
Project-Specific Build Checks	47
Build Check Output.....	47
OWASP Dependency Check Vulnerability Detection.....	48
The Dependency Check Utility.....	48
Running the Dependency Check Utility.....	48
The suppressions.xml File	49
Updating the Dependency Check Utility.....	49
Further Information on the Dependency Check Utility	50

Services Standard Build Overview

A build process is critical to the smooth deployment to production of a configured IdentityIQ environment. A build process helps streamline the process of promoting an IdentityIQ installation's configuration objects through the development, test and production environments so that all three contain the same custom objects like applications, rules, task definitions and identity mappings. It also allows for new custom objects and custom Java code to be integrated into IdentityIQ with a simple, manageable set of commands that can be easily automated.

The Services Standard Build (SSB) is a set of artifacts developed by the SailPoint Services team to support the build process for IdentityIQ deployments. These tools were designed with the following goals in mind:

- Automate effort of generating deployments for various environments such as development, testing, UAT, and production.
- Reduce time frame for new team members to become familiar with project structure and customizations.
- Reduce likelihood of errors due to improper deployment of patches, fixes, and configurations.
- Accelerate the software development process with useful methods and tools that make configuring IdentityIQ more efficient.
- Enable the SailPoint support team to quickly replicate IdentityIQ environments.
- Provide a build structure that is familiar to J2EE and servlet application developers that appeals to a broad audience.

The SSB tools should be configured directly after installing IdentityIQ in the first development environment for a project.

The SSB is a subset of what is known as the Services Standard Deployment (SSD). The SSB can be downloaded as a standalone build tool, but downloading the SSD will incorporate all elements of the SSB with some additional artifacts to help with deployment, known as the Services Standard Frameworks (SSF), Services Standard Test (SST) and Services Standard Performance (SSP). Configuration and use of the larger SSD and these other components is available on Compass but is outside the scope of this document.

SSB Components

SSB scripts (`build.xml`, `scripts/build.dev.xml`, etc.) utilize Apache Ant 1.8.2, along with `ant-contrib 1.0b3` and `catalina-ant` (for Tomcat 7.x).

Apache Ant

This is the main build tool using XML documents as build instructions. You don't have to do anything special to start using Ant when using the SSB – it's bundled right along with the other SSB artifacts (in the `lib/ant` folder)!

See the project page at: <https://ant.apache.org/>. There is a user guide for 1.9.x and 1.10.x – these are largely the same content that applies to the 1.8.x version used in SSB.

Ant Contrib

This has extensions for Ant and custom Ant tasks (like `<if><then>` blocks).

See the project page for more information: <https://sourceforge.net/projects/ant-contrib/files/ant-contrib/>.

Catalina-Ant

If using Apache Tomcat, Ant has extensions to manage certain aspects of the servlet container - generally this will only be used by those with more advanced SSB requirements.

While this included jar is technically part of Tomcat 7, it should work with later versions of Tomcat. If there is any doubt, the `catalina-ant.jar` file can always be replaced with one for your version of Tomcat (`<tomcat install folder>/lib`).

See the project page for more details if appropriate (Tomcat 8.0 example link):

<https://tomcat.apache.org/tomcat-8.0-doc/api/org/apache/catalina/ant/>.

Custom Ant Tasks

There are also custom Ant task extensions created by SailPoint that are bundled with the SSB that support the build scripts. These are in `servicestools/sailpoint/services/tools/ant` and should not be modified (unless there is a specific reason to do so). For more information about custom Ant task development see: <https://ant.apache.org/manual/develop.html>.

Process Overview

Before beginning, please ensure you have the following:

1. Access to SailPoint's Compass (community) website: <https://community.sailpoint.com/>
2. Command-line access to your development and/or test servers.
 - a. These are the servers where IdentityIQ's servlet container (web application server) runs.
 - b. This may be JBoss, Tomcat, WebSphere, or WebLogic, depending on the environment.
 - c. Note that command line access to your production servers is only necessary if you will be installing IdentityIQ and migrating custom code to your production servers.
3. Ability to create a directory in the WEB-INF/bin folder of your Identity IQ installation directory on your development server.
4. Ability to stop and start your web application server (Tomcat, JBoss, WebLogic, WebSphere, etc.).
5. Ability to copy a directory from your development server to your test and/or production server.

The steps you will perform to complete this process are as follows:

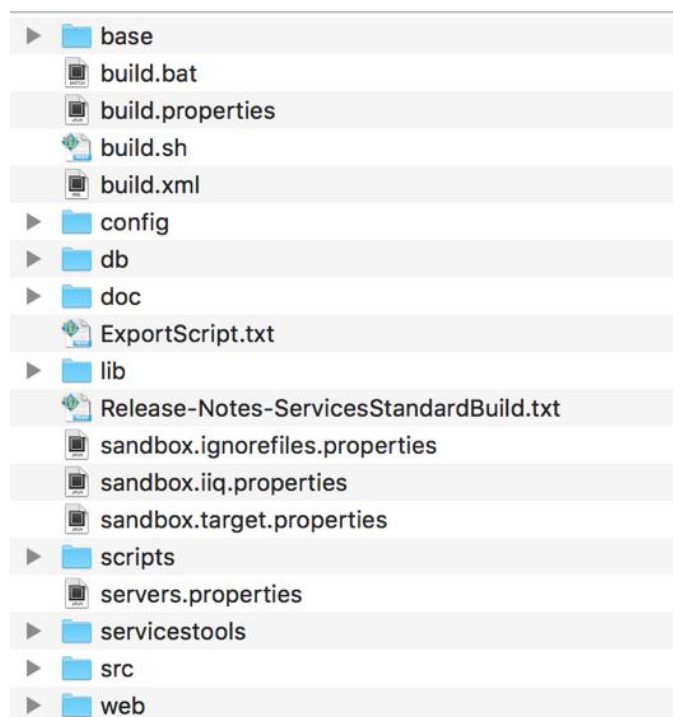
1. Download the Services Standard Build from Compass.
2. Export the custom objects from your development environment.
 - a. If this is a new IdentityIQ installation, there won't be any objects to export.
3. Set up the directory structure of the build.
4. Configure the build.
5. Run the build command.

Downloading the Services Standard Build

First, download the latest version of the standalone Services Standard Build or the full Services Standard Deployment from Compass.

Download the zip file with the latest version and unzip it into a file directory accessible to the development environment. This will create the base build structure. Make note of the directory where you have un-zipped the services standard build files; this directory will be called the “SSB Install Directory” throughout the remainder of this guide, and you will return to it repeatedly throughout the build process.

Folder Structure



This is the high-level folder structure of the build. The top-level directories should not be modified, though objects will be placed into these folders, either directly or in subfolders, to be used in the build process.

- **base** - Contains binaries distributed by SailPoint. You can download these from Compass.
 - **efix** - Contains any efix archives sorted by directory name where the directory name follows the naming convention `<version><patchlevel>`. If there is no patch level it will just have the version number. Because efix solutions only work with the specific product version they were designed for, you must make a unique directory for each version and patch level you want to build against. If a properly named efix directory is not found, the build will generate one. Efix files in .jar and .zip formats are supported. Efixes will be expanded in order of the creation timestamp on the zip or jar file; files with an earlier

creation timestamp will be expanded first. In the case of filesystems that do not have creation timestamps (Unix/Linux), the last modified timestamp will be used to define the order of expansion.

- **ga** - Contains the SailPoint GA release binary. You can have as many GA release binaries as you want to build against and the appropriate one will be selected using the values you set in the `build.properties` file.

Example: `/base/ga/identityiq-7.1.zip`

- **patch** - Contains the SailPoint patch binaries. You can have as many patch binaries as you want to build against and the appropriate patch will be selected using the values you set in the `build.properties` file.

Example: `/base/patch/identityiq-7.1p1.jar`

- **config** - Contains all your custom XML configuration objects sorted by folders where each sub directory is named by the type of top level SailPoint object it holds. In the provided example `Application`, `Rule`, `TaskDefinition` and `TaskSchedule` directories are shown. In general, as you customize more object types, you should add a directory to contain that object. While writing code, try to make the separation of object types as granular as possible such that it is easy to view all objects of a particular type. For example, instead of inserting a rule directly into a `TaskDefinition`, a reference to that rule should be created and the Rule itself would live in its own file in the Rule directory. The idea is to separate and encapsulate.
 - **Note:** While we recommend there just be objects in directories named for that object type (`Application`, `Bundle`, etc), there is nothing special about the directory names under the `config` directory. All files under `config` whose names end with a `.xml` suffix will be transformed through the build and tokenization and prepared for import into IdentityIQ. Files with other kinds of name extensions (`.txt`, `.old`, etc.) under `config` are ignored by the build process.
- **db** - Contains customized database scripts.
- **lib** - Contains libraries used by the build process. It contains Java code the Ant build scripts use, but it does not get added to your installation of IdentityIQ. Do not put additional jars here. Put them in the `web/WEB-INF/lib` directory.
- **scripts** - Except for the master `build.xml` file in the root directory, all other build files are contained in this directory. Shipped and supported build files are read-only and follow the name convention `build.*.xml`. If you customize the build process you must declare your customizations in build files that follow the naming convention `build.custom.*.xml`.
 - Three example scripts are provided illustrate how to extend the build process with site-specific custom scripts.
 - `scripts/example.build.custom.Extend-idAttrs.xml` (Configure extended searchable Identity attributes using the `ExtendedPropertyAccessor` class)
 - `scripts/example.build.custom.Modify-WEB-XML.xml` (Example of generic replacement of text in the `web.xml` file)
 - `scripts/example.build.custom.modify-web_xml_timeout.xml` (Modify the timeout value in `web.xml`)
 - Custom Ant scripts can inject their own site-specific logic in one of three places:

- The `clean` target, which allows the custom Ant script take whatever actions are necessary when resetting the builds to a clean or blank state.
 - The `post.expansion.hook` target, which allows the custom Ant script to implement site-specific logic after the build has expanded the stock IdentityIQ war file into the `build/extract` directory. This is an opportunity to transform files or alter what will end up in the finished `.war` file.
 - The `post.war.hook` target, which allows the custom script to take any action after the war file has been zipped together into a single file. This is commonly used for automated copying or deployment of the war file to a file server or repository.
- The example scripts provide guidance in their comments for readers interested in using them as templates to create their own site-specific build script functionality. They will not execute during the build process unless their names are changed to the `build.custom.*.xml` format.
- Readers interested in learning more about how Ant works are encouraged to review Apache Ant documentation: (<http://ant.apache.org/manual/>).
- **servicetools** - Contains the source code and an Ant project to build the `services-tools.jar` which is placed in the `lib` directory of the build. Code compiled and placed into the `services-tools.jar` is responsible for creating `sp.init-custom.xml`. Calling `import sp.init-custom.xml` from the `iiq` console is an additional way to push custom objects from your `<SSB Install Directory>/config` folder into your IdentityIQ database.
- **src** - Contains all your custom Java files. Note this Java will be compiled and placed in a jar file, which will be placed in the main IdentityIQ installation's `WEB-INF/lib` directory. It will be named based on the `customer` property in `build.properties`. The jar will become `identityiqCustomizations.customer.jar`. You should NOT "clone and own" SailPoint-shipped classes in this area. Since they will be placed in the `classpath` at the same level as the shipped classes, you may get behavior you do not expect. If you absolutely must modify a core class, you will have to define a `build.custom.*.xml` file to handle layout of these files as you are effectively defining your own `efix`. By default, the SSB will not acknowledge with this practice; it is discouraged.
- **web** - Contains content that will be directly overlaid on the IdentityIQ folder structure. Examples include: custom graphics/branding, `xhtml`, `jsp`, custom message catalogs, and additional jar libraries. Under `web` you will need to create the folder structure for the location where these files are normally stored. For information on custom branding for your enterprise, go here: <https://community.sailpoint.com/docs/DOC-7952>.
 - Example: to include custom changes to the Hibernate XML configuration file for identity extended attributes, put your customized version of `IdentityExtended.hbm.xml` in this directory nested in the full directory path: `web/WEB-INF/classes/sailpoint/object/IdentityExtended.hbm.xml`.

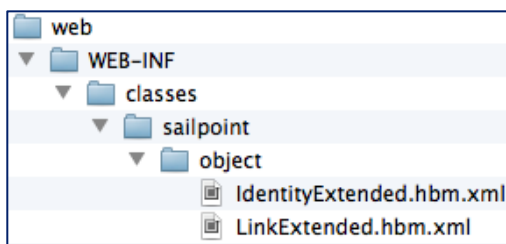


Figure 1 - Example of using "web" in the SSB folder structure

.keep Files

There are `.keep` files in several areas of the SSB folder structure. These are to preserve an empty folder structure if git is used for source control (which is increasingly common). While SVN preserves empty folders upon a check-in, git does not. Thus, a `.keep` file is a common way to make an empty folder a trackable object in git.

The `.keep` files are stripped out of `build` directory of the SSB project so they are not deployed to the web application server.

If you are not using git, these placeholder files can be removed from the core folders, but leaving them does no harm.

Exporting Custom Objects

This section assumes that IdentityIQ has been successfully installed into a development environment and that object definitions (e.g., applications, rules) have been created. If IdentityIQ has not been installed in at least your development environment, please do this first. If there are no custom object definitions to export now, skip this step and add them to the build's `<SSB install directory>\config` folder as they are created. No out of the box objects need to be added to your build directory; they are all added to the IdentityIQ database when running `import init.xml`, and if implementing the Lifecycle Manager functionality of IdentityIQ, `import init-lcm.xml`. However, if you change an out-of-the-box object (ObjectConfig-"Identity" is an example of a common out of the box object that changes when configuring identity mappings), this *does* need to be added to your build's config folder. This will ensure those changes migrate from environment to environment.

For further information on the object types that should be managed in your build, see the "**Best Practices: Deployment, Migration, Upgrade, and Artifact Management**" document on Compass here: <https://community.sailpoint.com/docs/DOC-2264>.

Note that there are several ways to export XML objects to the filesystem. The SailPoint Services team now recommends using the Object Exporter task (available in the SSD or separately on Compass) and/or the IdentityIQ Deployment Accelerator (also on Compass). However, the information below covers the Export Script, an older method of exporting XML objects which is detailed here for reference.

Running the Export Script

The SSB includes an export script (called Export Script.txt), which tells IdentityIQ to export some of the most common object classes. It exports all objects of each object type into a separate file per object type. For example, one line of the export script is `export -clean exports/CurrentApplicationExported.xml Application`. This line exports all the Application objects into a file called `CurrentApplicationExported.xml`. These objects must be exported from the development environment and included in the build directory tree to be included in the build process. (Note: The `-clean` argument will tell the exporter to strip the object of all Hibernate-generated IDs. This is important for porting objects between environments)

Use the `iiq console` command line utility to see the configuration objects your environment has by type:

1. Navigate to the `WEB-INF\bin` folder within your IdentityIQ installation directory from a command prompt. Enter the command `iiq console` once inside this directory.
2. Enter the command `list` to see all the object types or classes.
3. Enter `list <objectType>` to see all the objects of that type in your environment.

```
> list Application
Name
-----
Active_Directory
AdminsApp
Composite_ERP_Global_App_Users
Composite_ERP_Global_DB
Composite_ERP_Global_Platform
ERP_Global
HR_Contractors
HR_Employees
NBA Active Directory
Oracle_DB_oasis
Procurement_System
RACF1
RealADWithDemoData
RealLDAPWithDemoData
Schooner Active Directory
testInstancesApplication
>
```

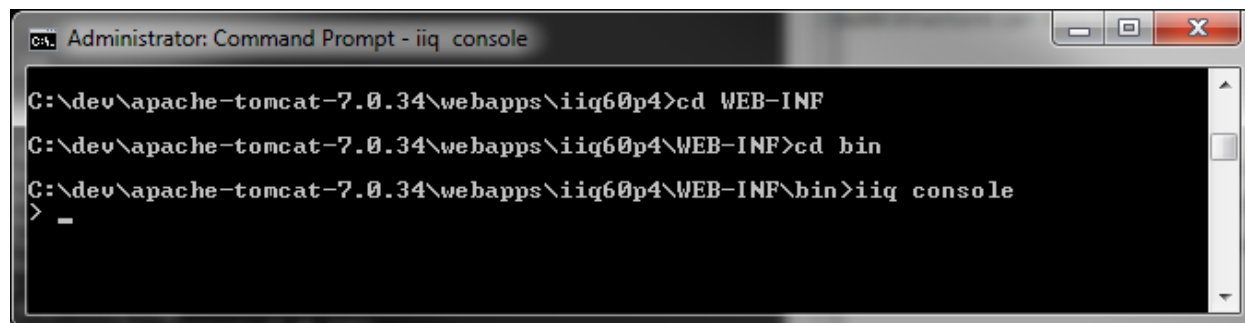
If your environment has configuration object types not covered in the object classes listed in the export script, edit the file to add more export commands, following the syntax of the provided lines:

```
export -clean exports/CurrentObjectClassNameExported.xml <ObjectClassName>
```

```
export -clean=id,created,modified exports/CurrentActivityDataSourceExported.xml ActivityDataSource
export -clean=id,created,modified exports/CurrentApplicationExported.xml Application
export -clean=id,created,modified exports/CurrentApplicationScorecardExported.xml ApplicationScorecard
export -clean=id,created,modified exports/CurrentAuditConfigExported.xml AuditConfig
export -clean=id,created,modified exports/CurrentAuthenticationQuestionExported.xml AuthenticationQuestion
export -clean=id,created,modified exports/CurrentBundleExported.xml Bundle
export -clean=id,created,modified exports/CurrentCapabilityExported.xml Capability
```

Copy the `ExportScript.txt` from the `<SSB install directory>` directory you unzipped earlier. Paste this text file into the `WEB-INF\bin` folder of your IdentityIQ installation directory. Also, create a folder called `exports` in the `WEB-INF\bin` folder.

Navigate back to the `WEB-INF\bin` folder within your IdentityIQ installation directory from a command prompt. Launch the console by entering the command `iiq console`.



```

Administrator: Command Prompt - iiq console
C:\dev\apache-tomcat-7.0.34\webapps\iiq60p4>cd WEB-INF
C:\dev\apache-tomcat-7.0.34\webapps\iiq60p4\WEB-INF>cd bin
C:\dev\apache-tomcat-7.0.34\webapps\iiq60p4\WEB-INF\bin>iiq console
>

```

When you see the `>` prompt, enter the command `source ExportScript.txt`. This will run the export script and export all your environment's configuration objects into the exports folder you just created.

```

TaskSchedule:
53f3c9712fe44c8297c1bb8dd66d45d6
8f19d525b3cb4f3c9b3ec848c27f03b8
b17e67600cf045b19568ef74cd741b31
Check expired mitigations daily
Check expired work items daily
Manager Certification [DATE] [4/1/13 1:55 PM]
Manager Certification [DATE] [4/3/13 1:54 PM]
Manager Certification [DATE] [4/3/13 2:09 PM]
Manager Certification [DATE] [4/3/13 2:39 PM]
Manager Certification [DATE] [4/4/13 10:25 AM]
Manager Certification [DATE] [4/4/13 11:59 AM]
Manager Certification [DATE] [4/4/13 12:04 PM]
Perform Identity Request Maintenance
Perform maintenance
TimePeriod:
first_quarter
second_quarter
third_quarter
fourth_quarter
holidays
weekdays
weekends
office_hours
non_office_hours
UIConfig:
UIConfig
Workflow:
Check Status of queued items
Do Manual Actions
Do Provisioning Forms
Entitlement Update
Identity Correlation
Identity Refresh
Identity Update
Password Intercept
Provision with retries
Role Modeler - Impact Analysis
Role Modeler - Owner Approval
Scheduled Assignment
Scheduled Role Activation
>

```







Many installations choose to split the export files into multiple files, storing each individual object in its own XML file. This practice is recommended, but not required. This makes it easier in the future to track exactly which objects have been changed between releases.

There is a Perl script on Compass that will perform the object separation. It is located here: <https://community.sailpoint.com/docs/DOC-2103>.

To split the objects up manually, copy an object's entire definition into a separate file, wrapped in the following header, opening, and closing tags:

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE sailpoint PUBLIC "sailpoint.dtd" "sailpoint.dtd">
<sailpoint>
<Put Object Definition Here>
</sailpoint>
```

Place each xml object into its respective class folder. The recommended naming convention for each of these object files is `ObjectType-Name.xml`. For example, `CurrentApplicationExported.xml` would be split into `Application-ActiveDirectory.xml` and `Application-PeopleSoft.xml`, etc.

Name	Date modified	Type	Size
 Application-Active_Directory.xml	3/29/2013 10:56 AM	XML Document	1 KB
 Application-AD.xml	3/29/2013 10:56 AM	XML Document	21 KB
 Application-eDirectory.xml	3/29/2013 10:56 AM	XML Document	8 KB
 Application-Infinium.xml	3/29/2013 10:56 AM	XML Document	6 KB
 Application-Peoplesoft.xml	3/29/2013 10:56 AM	XML Document	5 KB
 Application-SAS.xml	3/29/2013 10:56 AM	XML Document	5 KB

Build Structure Set-up

Configuration Objects

For the build process, all your environment's configuration objects should be placed into the `<SSB install directory>config` directory. Inside of this config folder, create a folder for every object class you exported. Folders for some objects -- `Application`, `LocalizedAttribute`, `Rule`, `TaskDefinition`, and `TaskSchedule` already exist as examples.

Place your exported xml files into their respective folders. For example, place the exported `Application` files into the `<SSB install directory>\config\Application` folder.

IdentityIQ Product Files

The build process will rebuild IdentityIQ for deployment into the target environment, merging the product zip files, patch jar files, and your custom artifacts. So next, you must put the desired product version zip files and patch jar files into the build directory tree.

Copy the zip file for the IdentityIQ version you are using into the `<SSB install directory>\base\ga` folder. Zip files can be downloaded from Compass if needed. **NOTE:** Multiple IdentityIQ zip files can coexist in this directory; a variable in the `build.properties` file for each environment determines which .zip file the build process will use.

If you are running a patched version of IdentityIQ, place the patch .jar file for your installation into the `<SSB install directory>\base\patch` folder. Again, multiple patch jar files can coexist in this directory and the `build.properties` file specifies which to use in the build (with the `IIQPatchLevel` variable). All patch .jar files can be downloaded from Compass as well.

If you have any efixes for your current patch, be certain to copy those to an appropriate efix directory and remember to check them into your revision control system if you are using one on your project.

Plugins

The SailPoint Plugin Framework is an extension framework model for IdentityIQ which enables third parties to develop rich application and service-level enhancements to the core SailPoint platform. For supported versions of IdentityIQ (7.1 and higher), plugins may be added to the build so that they will be built and/or automatically installed or uninstalled.

Build/Compilation of Plugins

The SSB build process can build and compile plugins automatically from the plugin source code. This requires that the plugins are placed under the `pluginsrc` folder at the root of the SSB, under a subfolder named for each plugin. In addition, the components of the plugin must be located in specific subfolders as shown in the table below.

Subfolder	Description
<code>pluginsrc/<PluginName>/db</code>	Contains the database scripts for the plugin (within <code>install</code> , <code>uninstall</code> and <code>upgrade</code> subfolders)
<code>pluginsrc/<PluginName>/import</code>	Contains the XML artifacts to be imported
<code>pluginsrc/<PluginName>/lib</code>	Contains any extra jar files that will ship with the plugin
<code>pluginsrc/<PluginName>/src</code>	Contains the source code for the plugin (in package subfolders)
<code>pluginsrc/<PluginName>/ui</code>	Contains the UI elements of the plugin (such as images, CSS files, HTML templates, and JavaScript)
<code>pluginsrc/<PluginName>/manifest.xml</code>	Mandatory file that defines plugin parameters

For more information on each of these components, please refer to the Plugin Developer Guide for IdentityIQ at <https://community.sailpoint.com/docs/DOC-7562>.

Plugins configured correctly under the `pluginsrc` folder will be built and compiled by the SSB. When building plugins with the SSB there is no need for the separate `build.xml` or `build.properties` files described in the Developer Guide.

Plugins will be built to the `build/plugins` folder when the main build is executed. The plugin zip file will be located in the `build/plugins/<PluginName>/dist` folder. It will also be copied to the `web/plugins/system/SSB/install` folder in the IdentityIQ build for automatic deployment (see below).

Automatic Deployment of Plugins

Automatic deployment of plugins relies on the presence of a `ServiceDefinition.xml` file to be imported with the build, and a `Jar` file which contains a service that manages plugin installation and removal. The files are in the following locations in the SSB:

```
config/ServiceDefinition/SSB_PluginImporterService.xml
web/WEB-INF/lib/ssb-plugin-importer.jar
```

In addition, the `deployPluginImporter` property in the `build.properties` file must be set to `true`.

If the plugin is not being compiled as part of the build process (see above) and you already have a plugin packaged in a zip archive file, you can automatically deploy it by placing the zip file in the correct location. If the plugin is being compiled by the build process, the zip file will automatically be created and deployed.

To have IdentityIQ install a plugin, place the plugin archive in the `web/plugins/system/SSB/install` folder of your SSB build. The following points apply for installing plugins in this way:

- Installation of any plugins in the `install` folder of the deployed build will be attempted on server start and thereafter once per day
- If the plugin is already installed it will not be reinstalled
- If the version of the plugin in the `install` folder is newer than the existing installed plugin it will be upgraded.

To have IdentityIQ uninstall a plugin, place the plugin archive that matches the installed plugin version in the `web/plugins/system/SSB/uninstall` folder of your SSB build. The following points apply for uninstalling plugins in this way:

- An attempt will be made to uninstall any plugins in the `uninstall` folder of the deployed build on server start and thereafter once per day
- If the plugin is not currently installed it will be ignored
- If the installed plugin is a different version than the plugin present in the `uninstall` folder the plugin will not be uninstalled.

The frequency at which the `install` and `uninstall` folders are searched for plugins can be varied by modifying the number of seconds defined in the `interval` property of the `PluginImporter` `ServiceDefinition` object.

JDBC Drivers

A common practice with any IdentityIQ deployment is to update the JDBC driver used specific to your database management system. This can help to avoid issues with performance and with vulnerabilities associated with outdated versions of the driver. A guide on Compass outlines this procedure:

<https://community.sailpoint.com/docs/DOC-4111>.

Note when a JDBC driver is put in the SSB project folder area `web/WEB-INF/lib`, the developer should check to remove the out-of-the-box JDBC driver by having the older jar deleted at build. This will ensure that a deploy includes only the latest JDBC jar specific to your environment.

First, get your updated JDBC driver and place it in the `web/WEB-INF/lib` area of the SSB project. For SQL Server, this might be `sqljdbc42.jar`. Run a `build clean main` target set and check the `build/extract/WEB-INF/lib` folder for legacy jar files for your database system. In this example, that may be `sqljdbc4.jar` (the default SQL Server driver that comes with IdentityIQ 7.0). Take note of the jar file (generally is only one) and adjust the `main` target in `build.xml`.

Add a line to the `main` target (near top of target, there are a few of them already) like this:

```
<delete file="{build.web-inf.lib}/sqljdbc4.jar"/>
```

In this example `sqljdbc4.jar` was used – each installation may differ. This is one exception where modifying the default SSB build file makes sense – usually the default targets and build files should not be modified.

Build Configuration

Configuring the build.properties file

The `build.properties` file is a crucial configuration file that specifies many important configuration arguments, like the version of IdentityIQ you are running, the Customer name, and the path to your IdentityIQ installation. Without this information, the build cannot run successfully.

Now configure the `build.properties` file found in the `<SSB install directory>`. Use your favorite text editor to edit this file.

```
#required properties
IIQVersion=7.1
IIQPatchLevel=
#application server iiq home
customer=AcmeBank
jdk.home.1.6=c:/Sun/Jdk
runCustomScripts=false

#dev properties
#make sure any app server specific env variables,like CATALINA_HOME for tomcat, are set on your system
IIQHome=c:/dev/tomcat/webapps/iiq
application.server.host=localhost
application.server.port=8080
application.server.start=c:/dev/tomcat/bin/startup.bat
application.server.stop=c:/dev/tomcat/bin/shutdown.bat
db.url=jdbc:mysql://localhost?useServerPrepStmts=true&tinyInt1isBit=true&useUnicode=true&characterEncoding=utf8
db.userid=root
db.password=password
db.driver=com.mysql.jdbc.Driver
iiq.path=/iiq/login.jsf
#type must be db2,mysql,oracle,sqlserver
db.type=mysql
db.name=identityiq63p

#information on the user account that will be created using the DB script
db.userName=ssbuser
db.userPassword=ssbpass

# sqlserver has an additional item created in the script for the Login name, which is separate from the user name. Specify that here
db.sqlserver.loginName=sqlLoginName

# db2 requires a separate DB name for its scripts and file/bufferpool. Specify those here
db.db2.databaseName=db2dbName
db.db2.bufferpool=db2bufferpool
db.db2.tableSpaceName=db2tableSpaceName

# For the createdb/dropdb scripts and Oracle, we have the option to uncomment the lines that create
# the tablespace and user, as well as removing them. These variables need to be set to enable that
db.oracle.createUser=true
db.oracle.createTableSpace=true
db.oracle.tableSpaceName=ssbtest
db.oracle.tableSpacePath=/home/oracle/app/oracle/oradata/ssbtest.dbf
db.oracle.useFastDropScript=false

installJavaMelody=true
#set this to true if you make test scripts that call targets with user warnings like drop db etc.
override.safety.prompts=false
#for export, the original install date string that we can use to determine new or changed objects
installDate=4/25/12 14:48:58 PM CDT

#tomcat properties. You only need to set these if you have a CATALINA_HOME env var set
#NOTE: for tomcat 7+ the manager url is /manager/text for 6 and lower its just /manager
#You will also need to setup a manager-script user in your tomcat user config, check out
#http://tomcat.apache.org/tomcat-7.0-doc/manager-howto.html#Executing_Manager_Commands_With_Ant
manager.url=http://localhost:8080/manager/text
manager.login=tomcat
manager.pw=tomcat
tomcat.home=c:/dev/tomcat

# If Lifecycle Manager is required you can have it included in the init.xml by setting usingLcm to true (i.e. usingLcm=true)
usingLcm=false

# Username and encrypted password used by importdynamic to access the console.
# This can be removed, in which case the user will be prompted for credentials.
console_user=spadmin
console_pass=1:p+qvPB04Rig8PY1NWbr3Zg==

# Add loggers found in BeanShell and Java code to log4j.properties
updateLog4jLoggers=false
```

Set properties in the `build.properties` as described in the table below.

Variable	Description	Required?
IIQVersion	Specify the base version of IdentityIQ that you are building, e.g. 6.0, 6.1, 6.2, 6.3, 6.4, 7.0, 7.1	Yes
IIQPatchLevel	If you want to deploy a patch version, specify what level with pX syntax, e.g. p1 or p6. If you are deploying only the GA version, leave this blank.	No, only if deploying a patch version
IIQHome	The home directory of the IdentityIQ web application in your sandbox/development environment. When using the <code>deploy</code> build target, the <code>IIQHome</code> property tells the build where to deploy your custom IdentityIQ installation.	No, only when using <code>deploy</code> target
customer	The name of the client or project phase. The build will create a <code>.jar</code> file, compiling all <code>.java</code> code in the build's <code>src</code> folder and name that jar <code>identityIqCustomizations.Customer.jar</code> .	Yes
jdk.home	The path on your system to the Java Development Kit (jdk) you want to use to compile any custom Java code you may have developed as part of your IdentityIQ configuration. As with all system paths, if there are spaces in your jdk path, put the entire path in double quotes. In lieu of this, you can set the <code>JAVA_HOME</code> environment variable for your OS.	No
runCustomScripts	(true/false) Generally, the default SSB build scripts are not meant to be modified directly. The main build has two hook points after file layout and after war creation where you can execute customized build scripts. This flag indicates if these customizations should be executed.	Yes, leave as false if unsure
runCodeChecks	If set to true, checks as defined in the Build Checks section of this document are performed.	No
codeCheck.namingConvention	Defines a naming convention used by the checks described in the Build Checks section of this document.	No
application.server.host	The IP address of your application server in your sandbox/development environment	No, only when using <code>cycle</code> or <code>importcycle</code> build targets
application.server.port	The port the application server is running on. For example, 8080 is the Tomcat default.	No, only when using <code>cycle</code> or <code>importcycle</code> build targets

application.server.start	Script to start the application server. Since there are so many different application servers we leave it to you to write a script that starts and stops the server, sets up JVM parameters etc. Many application servers already ship with these but you can specify which ones you want to use here. This script (and the stop script below) is used in development targets that include steps to cycle the application server for you.	No, only when using <code>cycle</code> , <code>importcycle</code> , <code>up</code> , or <code>down</code> targets
application.server.stop	Script to stop the application server.	No, only when using <code>cycle</code> , <code>importcycle</code> , <code>up</code> , or <code>down</code> targets
db.url	The JDBC URL to your local database.	No, only when using targets <code>createdb</code> , <code>dropdb</code> , <code>extenddb</code> , <code>initial-build</code> , etc.
db.userid	Database user with create and drop schema privileges (e.g. <code>root</code> on MySQL). NOTE: Supply this parameter only for low-risk, non-production environments (e.g. <code>sandbox/development</code>), as it is not designed for production environment use at this time.	No, only when using targets <code>createdb</code> , <code>dropdb</code> , <code>extenddb</code> , <code>initial-build</code> , etc.
db.password	The password for the root DB user. Not supported as an IdentityIQ-encrypted string at this time. NOTE: Supply this parameter only for low-risk, non-production environments (e.g. <code>sandbox/development</code>), as it is not designed for production environment use at this time.	No, only when using targets <code>createdb</code> , <code>dropdb</code> , <code>extenddb</code> , <code>initial-build</code> , etc.
db.driver	The class of the JDBC driver to use for SQL connections. This is the same value you would put in your <code>iiq.properties</code> file, as instructed in the IdentityIQ Installation Guide.	No, only when using targets <code>createdb</code> , <code>dropdb</code> , <code>extenddb</code> , <code>initial-build</code> , etc.
iiq.path	The installation directory within the application server directory of the IdentityIQ application. Usually <code>/iiq</code> or <code>/identityiq</code>	Yes
db.type	One of these values: <code>db2</code> , <code>mysql</code> , <code>oracle</code> , <code>sqlserver</code> ; used to pick which database scripts to run	No, only when using targets <code>createdb</code> , <code>dropdb</code> , <code>extenddb</code> , <code>initial-build</code> , etc.
db.name	Name of the IdentityIQ database	No, only when using targets <code>createdb</code> , <code>dropdb</code> , <code>extenddb</code> , <code>initial-build</code> , etc.

db.userName	Name of user account that will be created using the DB script	No, only when using targets <code>createdb</code> , <code>dropdb</code> , <code>extenddb</code> , <code>initial-build</code> , etc.
db.userPassword	Password of user account that will be created using the DB script. Not supported as an IdentityIQ-encrypted string at this time. NOTE: Supply this parameter only for low-risk, non-production environments (e.g. sandbox/development), as it is not designed for production environment use at this time.	No, only when using targets <code>createdb</code> , <code>dropdb</code> , <code>extenddb</code> , <code>initial-build</code> , etc.
db.sqlserver.checkpolicy	SQL Server setting that defines whether Windows password policy should be checked when creating a login for SQL authentication. Default is <code>off</code> .	No, only when using targets <code>createdb</code> , <code>dropdb</code> , <code>extenddb</code> , <code>initial-build</code> , etc.
db.sqlserver.loginName plugin.db.sqlserver.loginName	SQL Server has an additional item created in the script for the Login name, which is separate from the user name. Specify that here. IdentityIQ 7.1 and SSB v4 added the <code>plugin.db.sqlserver.loginName</code> for the plugin DB.	No, only when using targets <code>createdb</code> , <code>dropdb</code> , <code>extenddb</code> , <code>initial-build</code> , etc.
db.db2.databaseName db.db2.bufferpool db.db2.tableSpaceName	DB2 requires a separate database name and file/bufferpool for its scripts. Specify those values here.	No, only when using targets <code>createdb</code> , <code>dropdb</code> , <code>extenddb</code> , <code>initial-build</code> , etc.
db.oracle.createUser db.oracle.createTableSpace db.oracle.tableSpaceName db.oracle.tableSpacePath db.oracle.useFastDropScript	For the <code>createdb/dropdb</code> scripts for Oracle, we have the option to uncomment the lines that create the tablespace and user, as well as removing them. These variables need to be set to enable that.	No, only when using targets <code>createdb</code> , <code>dropdb</code> , <code>extenddb</code> , <code>initial-build</code> , etc.
installJavaMelody	If using JavaMelody, set this to true to gather SQL statistics in Oracle	No, only when using targets <code>createdb</code> , <code>dropdb</code> , <code>extenddb</code> , <code>initial-build</code> , etc.
override.safety.prompts	Certain dangerous build targets like <code>dropdb</code> will prompt the user for confirmation before executing. If you are using the build to make test cases you may want to turn off these prompts.	Yes, leave as false if unsure
installDate	For the <code>export</code> target, the original install date string that we can use to determine new or changed objects.	No, only if using the <code>export</code> target
manager.url	URL to the tomcat manager script interface; Prior to Tomcat Version 6 the URL is usually <code>/manager</code> but post Version 6 it is <code>/manager/text</code> .	Only if deploying using Tomcat application server

manager.login	A user who has the <code>manager-script</code> role in the Tomcat manager application. For information on how to set this up check out: http://tomcat.apache.org/tomcat-7.0-doc/manager-howto.html#Executing_Manager_Commands_With_Ant	Only if deploying using Tomcat application server
manager.pw	The password for the above account.	Only if deploying using Tomcat application server
tomcat.home	Set this to the value of <code>CATALINA_HOME</code> you want to use when starting and stopping Tomcat	Only if deploying using Tomcat application server
usingLcm	If your implementation includes Lifecycle Manager, you can ensure that it is included in your project build by setting the <code>usingLcm</code> property to <code>true</code> . This will insure that <code>init-lcm.xml</code> is imported if the target <code>import-lcm</code> is called directly or indirectly.	Yes
console_user	Username used by <code>importdynamic</code> (and other targets utilizing import functionality) to access the console.	See section: Non-default spadmin password and importing artifacts
console_pass	Encrypted password used by <code>importdynamic</code> (and other targets utilizing import functionality) to access the console.	See section: Non-default spadmin password and importing artifacts
updateLog4jLoggers	If set to <code>true</code> , the <code>log4j.properties</code> file in <code>WEB-INF/classes</code> will be updated during the build process with a line for every logger that is found in BeanShell code in the XML files or in custom Java source code. These lines will be commented out. This helps during troubleshooting when a logger needs to be enabled but the name of the logger is not known without looking it up in the code. To enable a logger the appropriate line just needs to be uncommented in <code>log4j.properties</code> and set to the required log level before refreshing the logging configuration in IdentityIQ.	No
usingDbSchemaExtensions	This switch enables the <code>extenddb</code> target to be run. It would be enabled (<code>true</code>) if <code>IdentityExtended.hbm.xml</code> (or	Only if you plan to use the <code>extenddb</code> or <code>initial-build</code> targets

	similar object hbm.xml) was customized with named columns and placed in web/WEB-INF/classes/sailpoint AND ObjectConfig for a matching object was customized. Default is false.	
plugin.db.name plugin.db.userName plugin.db.userPassword	These mirror the “normal” db.name and similar settings. These were introduced in SSB v4 to handle the IdentityIQ 7.1 plugin DB. The password is not supported as an IIQ-encrypted string at this time. NOTE: Supply these parameters only for low-risk, non-production environments (e.g. sandbox/development), as they are not designed for production environment use at this time.	No, only when using targets createdb, dropdb, extenddb, initial-build, etc.
deployPluginImporter	If set to true and the version of IdentityIQ being deployed is 7.1 or higher, the PluginImporter ServiceDefinition object will be deployed, enabling plugins to be automatically installed from the filesystem.	No

Note that there are also some other variables in the `build.properties` file that start with `deploy`, such as `deploySSF`, `deployGenericImporter` and `deployObjectExporter`. These are only used in the full SSD to define which of the SSD components and tools should be deployed in the build. In the stand-alone SSB they are not used.

Supporting multiple platforms (Windows/Linux/Unix) for different environments

If your installation uses different operating systems for different stages of IdentityIQ development – for example, Windows for sandboxes and Linux for Test and Production servers – you must configure multiple `build.properties` files.

The generic `build.properties` file described above loads the defaults for the build with respect to the path to Java binaries, IdentityIQ version and other details. These can be overridden on a per-server or per-environment basis by specifying another properties file with properties that just apply to one server or one environment. Each server or environment used in development and testing can override the settings in `build.properties` by using its own `<hostname>.build.properties` or `<environment>.build.properties` file. For example, if your host is named `sailsandbox` then the properties file unique to that server would be called `sailsandbox.build.properties`. Or if your environment (SPTARGET) is called `dev` you could have `dev.build.properties`. The server or environment’s properties file has exactly the same format and fields as the `build.properties` file described in the previous section and only has to specify the fields that it wants to override with values that are different from the default `build.properties` file’s values. If you are running a build on a

server that has its own server-specific version of `build.properties` for an environment that has its own environment-specific version, the server-specific values override the environment-specific values, which in turn override the generic `build.properties` file. If you have servers or environments with identical `build.properties`, do not create server-specific or environment-specific files. Put those values in `build.properties`. The build will recognize that there is no file specific to the server or environment, and will use `build.properties` as the default.

Note: Prior to version 4 of the SSB `build.properties.<environment>` or `build.properties.<hostname>` were supported. Note those naming conventions, while still supported, have been deprecated and may be removed in a future release of the SSB. This has been done to further standardize the naming convention of various SSB host- and environment-specific artifacts. This also means the `.properties` extension is recognized by various properties-aware editors.

Non-default spadmin password and importing artifacts

The default `spadmin` username and encrypted password are set using the `console_user` and `console_pass` properties in `build.properties`.

```
# Username and encrypted password used by importdynamic to access the console.
# This can be removed, in which case the user will be prompted for credentials.
console_user=spadmin
console_pass=1:p+qvPBo4Rig8PYlNWbr3Zg==
```

These can be used to inject credentials when targets using `console iiqBeans` are employed. This may involve several targets like: `import-custom`, `import-stock`, `import-lcm`, `import-all`, `importdynamic`, `deploy`, `importcycle`, etc.

Use the console command `iiq encrypt <password>` to get the encrypted value of your password to use here. Alternatively, the `console_user` and `console_pass` lines can be removed from `build.properties`, which will force the user to enter them each time `importdynamic` (or a similar target using import functionality) is run.

Setting up environment-specific properties files

The goal of the build process is to create a uniform Identity IQ distribution process for all environments (development, test, and production), but each environment will need many different parameters that are specific to that environment, such as login usernames, IP addresses, passwords, database connection strings etc. It is important that the build process can perform substitution of these values for each environment, as well as ensuring connection to the right IdentityIQ database and, where necessary, including or excluding files from an environment-specific build.

Configuring iiq.properties files

The `iiq.properties` file contains properties used by IdentityIQ for connecting to and interacting with its own database. Your build environment can specify different `iiq.properties` files for the build to use for deploying to each target environment. Create separate `<environment>.iiq.properties` files

for each environment by copying and editing the product `iiq.properties`, and place them in the build directory (wherever you unzipped the SSB).

1. Download the product image
2. Expand the `identityiq-<version>.zip`
3. Expand the `identityiq.war`
4. Copy the `iiq.properties` to the SSB `<environment>.iiq.properties` file.
 - a. For example Copy `WEB-INF/classes/iiq.properties` to `sandbox.iiq.properties`

For example, if your environments are `sandbox`, `test`, `UAT` and `prod`, you would have four files each containing the `iiq.properties` that know how to connect to the database server in that environment. This way you can support different properties for different environments, such as having a direct connection in `sandbox` and `test` while having a JNDI named connection in `UAT` and `production`. When creating these `<environment>.iiq.properties` files, use the `iiq.properties` file that ships with your IdentityIQ version's `.zip` file and edit as appropriate for the environment.

Example file names:

```
sandbox.iiq.properties
test.iiq.properties
UAT.iiq.properties
prod.iiq.properties
```

Example `test.iiq.properties` file (yours may differ due to IdentityIQ version changes):

```
##### iiq.properties #####
#
# (c) Copyright 2008 SailPoint Technologies, Inc., All Rights Reserved.
#
# This file contains configuration settings for IdentityIQ. For your unique
# environment, you will need to adjust the username and password properties on
# the dataSource below and uncomment the applicable database settings.
#

##### Data Source Properties #####
dataSource.maxWait=10000
dataSource.maxActive=50
dataSource.minIdle=5
#dataSource.minEvictableIdleTimeMillis=300000
#dataSource.maxOpenPreparedStatements=-1

dataSource.username=root
dataSource.password=root

##### MySQL 5 #####
## URL Format:
dataSource.url=jdbc:mysql://<host_name>:<port>/<dbname>?useServerPrepStmts=true&tinyIntlisBit=true&useUnicode=true&characterEncoding=utf8
dataSource.url=jdbc:mysql://localhost/identityiq?useServerPrepStmts=true&tinyIntlisBit=true&useUnicode=true&characterEncoding=utf8
dataSource.driverClassName=com.mysql.jdbc.Driver
sessionFactory.hibernateProperties.hibernate.dialect=org.hibernate.dialect.MySQL5InnoDBDialect
```

```
#
# Setting for the BSFManagerPool set on the ruleRunner
#
bsfManagerFactory.maxManagerReuse=100
bsfManagerPool.maxActive=30
bsfManagerPool.minEvictableIdleTimeMillis=900000
bsfManagerPool.timeBetweenEvictionRunsMillis=600000

##### Debug Settings #####

# Uncomment to send all SQL queries to std out. This provides a lot of output
# and slows down execution, so use it wisely.
#sessionFactory.hibernateProperties.hibernate.show_sql=true

# Hibernate Transaction Isolation Levels
# 1 = Read Uncommitted, 2 = Read Committed, 4 = Repeatable Read, 8 = Serializable
#sessionFactory.hibernateProperties.hibernate.connection.isolation=1
```

Configuring target.properties files

It is important to configure environment-specific properties files that the SSB can use to do token string replacements in the objects during the build process. The SSB will automatically look for tokenized strings in your custom configuration XML and substitute the appropriate values per environment. A `target.properties` file should be created for each environment, containing key/value pairs for token substitution during build time. The name of each `target.properties` file should be in the format `<environment>.target.properties`.

Examples:

```
sandbox.target.properties
test.target.properties
UAT.target.properties
prod.target.properties
```

Each file is just a list of key/value pairs. The build's convention is that the keys follow a `%%KEYNAME%%` pattern.

For example, you may have an Active Directory application configuration that looks like this:

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE sailpoint PUBLIC "sailpoint.dtd" "sailpoint.dtd">
<sailpoint>
<Application authoritative="true" connector="sailpoint.connector.ADLDAPConnector"
featuresString="AUTHENTICATE, MANAGER_LOOKUP, SEARCH, UNSTRUCTURED_TARGETS" name="AD"
profileClass="" type="Active Directory">
  <Attributes>
    <Map>
      <entry key="IQServiceHost" value="iqservicehost.example.com"/>
      <entry key="IQServicePort" value="5051"/>
      <entry key="password" value="2:omj3ooouHSFb7dIPItTjNigBCeZjxP+Vr9TewSXIbxs="/>
      <entry key="managerCorrelationFilter">
        <value>
          <Filter operation="EQ" property="DN" value="manager"/>
        </value>
      </entry>
    </Map>
  </Attributes>
</Application>
</sailpoint>
```

```

<entry key="user" value="productionADuser"/>
<entry key="groupHierarchyAttribute" value="memberOf"/>
<entry key="authorizationType" value="simple"/>
...

```

Note that the password has been encrypted using the `iiq encrypt` utility; you should always do this, especially if the password values are being stored in a properties or XML file that is part of a build stored in a location where it may be accessible by users who do not need to know it.

To support deploying the same XML artifact to multiple environments, you would substitute passwords, ports, etc. with keys that will go in your `<environment>.target.properties` file, so your application configuration file instead looks like this:

```

<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE sailpoint PUBLIC "sailpoint.dtd" "sailpoint.dtd">
<sailpoint>
<Application authoritative="true" connector="sailpoint.connector.ADLdapConnector"
featuresString="AUTHENTICATE, MANAGER_LOOKUP, SEARCH, UNSTRUCTURED_TARGETS" name="AD"
profileClass="" type="Active Directory">
  <Attributes>
    <Map>
      <entry key="IQServiceHost" value="%%AD_IQSERVICE_HOST%%"/>
      <entry key="IQServicePort" value="%%AD_IQSERVICE_PORT%%"/>
      <entry key="password" value="%%AD_PROXY_PASSWORD%%"/>
      <entry key="managerCorrelationFilter">
        <value>
          <Filter operation="EQ" property="DN" value="manager"/>
        </value>
      </entry>
      <entry key="user" value="%%AD_PROXY_USER%%"/>
      <entry key="groupHierarchyAttribute" value="memberOf"/>
      <entry key="authorizationType" value="simple"/>
    </Map>
  </Attributes>
</Application>
...

```

Then, for example, in the file `prod.target.properties` you would have:

```

%%AD_IQSERVICE_HOST%%=iqservicehost.example.com
%%AD_IQSERVICE_PORT%%=5051
%%AD_PROXY_USER%%=productionADuser
%%AD_PROXY_PASSWORD%%=2:omj3oouHSFb7dIPItTjNIgBCeZjxP+Vr9TewSXIIbxs=

```

... and so on for each of your environments.

If a token exists in an XML artifact but there is no corresponding token in the `target.properties` file, the build process will prompt the user for the value that should replace the token. The token will then be created in the `target.properties` file and its value replaced in the resulting XML file during the build.

Configuring “secret” target.properties files for storing sensitive token values

In the above example, a username and an encrypted password are stored in the `target.properties` file. Some customers may consider even the encrypted password to be too risky to store in a file on their version control repository or other location where it could be accessed by users who do not need to know this password. Other values may also be considered too sensitive to store in that location. By

using a “secret” target.properties file, the sensitive values can be stored in a separate file, and this file can then be excluded from the version control system and moved to a more secure location. A trusted user managing the build process can be given access to this file and can use it in a local copy of the SSD so that the sensitive tokens can be used when creating the build. The name of the file used for the “secret” target.properties file should be in the format <environment>.secret.target.properties and the values stored in it will be used to replace the tokens in the XML file in the same way as the regular target.properties file.

In the above example, the `%%AD_PROXY_USER%%` and `%%AD_PROXY_PASSWORD%%` tokens may be considered too sensitive to be stored in the main target.properties file and would be removed, so that the `prod.target.properties` file would only have these values for the AD application:

```
%%AD_IQSERVICE_HOST%%=iqservicehost.example.com
%%AD_IQSERVICE_PORT%%=5051
```

The sensitive tokens would then be stored in a file called `prod.target.secret.properties`:

```
%%AD_PROXY_USER%%=productionADuser
%%AD_PROXY_PASSWORD%%=2:omj3oouHSFb7dIPItTjNIgBCeZjxP+Vr9TewSXIbxs=
```

The build process will first attempt to replace tokens found in XML files using corresponding values found in the target.properties file, followed by any others found in the secret.target.properties file.

Configuring Subset Builds with includefiles.properties files

Depending on a customer’s preferred processes for deploying changes to IdentityIQ, it is sometimes useful to be able to define a ‘subset’ build, which only includes a specific set of XML files to import into an existing IdentityIQ environment, without deploying an entire build. This can be achieved by using an `includefiles.properties` file specifying the set of XML files to include in the subset build; all other XML files will be excluded from the subset build.

To enable subset builds as part of a regular SSB build, set the `buildSubset` property to `true` in the `build.properties` file:

```
buildSubset=true
```

Create an `includefiles.properties` file for each environment where you want to create a subset build. A sandbox example file is provided with the build (`sandbox.includefiles.properties`). You can copy, paste, and rename this file for all your environments as a template to get started.

See the provided file for an example of how to populate the list of files that will be included in each environment.

The file naming pattern is `<environment>.includefiles.properties`. For instance:
`prod.includefiles.properties` or `test.includefiles.properties`.

On running a build, a subfolder called `subset` will be created under the `build` folder. This will have a `WEB-INF/config/custom` folder structure where the subset XML files are located. There is also a `sp.init-custom.xml` file under `WEB-INF/config` that references the files as `ImportAction` lines so that when this file is imported, all the subset files will be imported. The folder structure should be copied to

an IdentityIQ server and overlaid over the existing `WEB-INF` folder in the IdentityIQ application, and the `sp.init-custom.xml` file imported.

For convenience of copying the subset to a server, a zip file of the subset folder structure is created under the `build/deploy` folder after running a build. This is named `identityiqSubset.zip`.

Note that the subset build is only useful for deploying specified XML objects to an existing IdentityIQ system and does not include any items that reside on the filesystem, such as IdentityIQ product files, custom Java code or branding. It is not a substitute for the full build process.

Configuring `ignorefiles.properties` files

The build process can be configured to define certain XML files that should be skipped during the import for a specific environment. Implement this by creating a text file for each environment where you want to skip the import of specific files. A sandbox example file is provided with the build (`sandbox.ignorefiles.properties`). You can copy, paste, and rename this file for all your environments as a template to get started.

See the provided file for an example of how to populate the list of files that can be ignored in each environment.

The file naming pattern is `<environment>.ignorefiles.properties`. For instance: `prod.ignorefiles.properties` OR `test.ignorefiles.properties`.

It is recommended to have an `ignorefiles.properties` file for each environment the SSB manages.

The SSF (Services Standard Framework) comes with a template ignore file called `ssf.ignorefiles.properties` that may contain important information (depending upon features used). Refer to the related SSF guide for more detail. Whatever the application of an ignore file, there is only one per environment, and it can contain a large number of items.

This feature has multiple uses. Below are a few examples:

- Configure different applications with the same name that contain entirely different XML contents and connector configurations in your development versus UAT and production environments
 - For example, you could use a JDBC application to simulate Active Directory in your sandbox environment but use a “proper” Active Directory application in your UAT and production environments
- Selectively load temporary or testing applications in your sandbox or development environments and not load those applications in your UAT or production environments
- Load quick link objects into certain environments and redact them from others, allowing customization of dashboard configurations for each environment
- Utilize different system configuration files across your environments by redacting different configuration files from each environment's build

Note: Care should be taken when choosing which files to ignore in specific environments. One goal of the SSB is to synchronize the configuration elements across development, UAT/test, and production installations of IdentityIQ. Using this feature intentionally creates configuration drift, and it should be thoughtfully managed.

Configuring `log4j.properties` files

Log4j is used for IdentityIQ logging, with the loggers and log levels defined in the `log4j.properties` file. It is possible to define a custom log4j file with a name in the format

`<environment>.log4j.properties` for each environment. This file should be at the root of the build and will be copied as `log4j.properties` to the `WEB-INF/classes` folder in the resulting build for a specific environment.

Note that if you also have the “updateLog4jLoggers” property set to “true” in `build.properties` (see “Configuring the `build.properties` file” above), the resulting `log4j.properties` will include the entries in the environment-specific `log4j.properties` file as well as commented-out loggers discovered from BeanShell code in XML artifacts or from Java source code in the build.

Configuring deployment of encryption keys for each environment

IdentityIQ can be configured to use site-specific encryption keys for encrypting and decrypting passwords, and this is considered best practice. This makes use of the IdentityIQ keystore, and with this feature enabled a password used on one site cannot be decrypted on another site without having the site-specific encryption keys. It is also recommended to use different encryption keys for each environment. Information on the keystore and its configuration can be found in the IdentityIQ Administration Guide, and additional details are available at <https://community.sailpoint.com/docs/DOC-2031>.

Once the keystore is configured for an IdentityIQ environment, the keystore files are stored in the following default locations on each server:

```
WEB-INF/classes/iiq.cfg  
WEB-INF/classes/iiq.dat
```

The SSB can manage deployment of keystore files only if they are stored in the default location. Some customers place the files in a location external to the IdentityIQ application and reference their location in the `iiq.properties` file. This can enhance security by limiting filesystem access to this location to specific trusted users, but the SSB will not be able to manage the keystore in these cases, although it will still be possible to specify the location of the files in the environment-specific `iiq.properties` file using the `keyStore.file` and `keyStore.passwordFile` properties (see the documentation).

It is also important to point out that if you choose to let the SSB manage deployment of the keystore files they will need to be stored in the build files on the filesystem. Usually, the build will be committed to a version control system repository, and this may result in the keystore files being available to users who should have access to the main build files but should not be given access to the keystore.

The decision to allow the SSB to manage the deployment of the keystore files rests with the customer, and if the access to the build can be protected it is a convenient way to push the keystore files out with the build. If this is not done, the keystore files will need to be stored in an external location or copied manually to the default location on each server after deploying a build.

To manage the deployment of the keystore files, prefix their names with ‘`<environment>.`’ and place them at the root of the build. For example:

```
prod.iiq.cfg  
prod.iiq.dat
```


The files will be deployed in the resulting build as:

```
WEB-INF/classes/iiq.cfg
WEB-INF/classes/iiq.dat
```

Setting the environment name for a build

After configuration of the environment-specific properties files, the build process needs to be told which environment to build for. This can be done by the use of the `SPTARGET` environment variable or alternatively by using a mapping in the `servers.properties` files.

Note that when both methods are employed, the `SPTARGET` environment variable “wins”. This is by design to provide on-the-fly flexibility.

Using the SPTARGET environment variable to specify the build environment

The `SPTARGET` environment variable will dictate which `<environment>.iiq.properties` file, `<environment>.target.properties` file, `<environment>.ignorefiles.properties` and (where configured) `<environment>.build.properties` file to use. Here is an example of using the `SPTARGET` environment variable on a Linux system to create a dev, test, and prod war file. See the next section for more details on executing the build.

```
./build.sh clean

export SPTARGET=dev
./build.sh war
mv identityiq.war identityiq-dev.war

export SPTARGET=test
./build.sh war
mv identityiq.war identityiq-test.war

export SPTARGET=prod
./build.sh war
mv identityiq.war identityiq-prod.war
```

Setting the target variables by editing servers.properties

If the `SPTARGET` environment variable is not set, the build process will attempt to find the environment name by using the `servers.properties` file, mapping the host where the build is being executed to an environment name. It is common for deployments to specify each environment’s hostnames in the `servers.properties` file to support running the build on every server in the environment. This tells the build which environment you want to use, which depends on the name of the server running the build. This is an example of a `servers.properties` file:

```
YOURMACHINENAME=sandbox
SAILPTAPP=prod
SAILPTDEV=dev
SAILPTTEST=test
```

Replace `YOURMACHINENAME` with your sandbox hostname, `SAILPTAPP` with your production hostname, etc. The build will detect the hostname of the machine on which you are running the build script, and apply `sandbox.target.properties` if you are running the build on your sandbox. There can be multiple hosts pointing to the same target (e.g. if you have 2 prod application servers, you might have 2 hostnames (1 per line) pointing to `prod`).

Note: The hostnames used in `servers.properties` are case-sensitive. To get the proper value, you need to check an environment variable for each host entry. For Windows, use the value exactly as specified by the `COMPUTERNAME` environment variable (e.g. `echo %COMPUTERNAME%`). For Linux/Unix/Mac, use the value exactly as specified by the `HOSTNAME` environment variable (e.g. `echo $HOSTNAME`).

Executing the Build

Once you have performed all the steps in the previous sections, you are ready to build. If you are using the `servers.properties` file to define the build environment name, copy your entire build structure and add it to your new environment. Otherwise, set the `SPTARGET` environment variable to the correct target environment name on the host that will run the build. This helps to ensure that environment-specific variables defined in `<environment>.iiq.properties`, `<environment>.target.properties` and `<environment>.ignorefiles.properties` will be added to your war file. Ensure your `build.properties` file is configured to match the environment.

A common approach to copying the entire build to a new server is to check in the entire build (and all the files, directories, and artifacts included with it) into a revision control system like SVN, CVS, Git or TFS. Tools like these can automate checking out the entire current copies of set of files onto new target servers. More basic installations sometimes simply "zip up" the entire set of artifacts and transfer one file to the new target server (e.g. a generated war file, as demonstrated later in this section). As a best practice, SailPoint strongly recommends using a revision control system if one is available. If existing source control is not available, git could be easily used to create a stand-alone, offline repository for tracking with no additional architecture.

To create a custom IdentityIQ war file (J2EE Web Application Archive) that can be deployed to a web application server such as Tomcat, you can perform the following steps.

- Open a Terminal or Command Prompt window
- Navigate to `<SSB install directory>\`
- Enter `build war`
- This will generate a deployable war file in your `<SSB install directory>\deploy` folder and you will receive a confirmation message like the one below:

```
war:
    [war] Building war:
    /home/workspace/SSB/build/deploy/identityiq.war
    [echo] A MD5 checksum was generated for this war file and
    placed in the war file directory. Keep this checksum to diagnose
    potential version issues

BUILD SUCCESSFUL
```

- Deploy this file to the target web application server. You may need to consult your application server’s deployment guide for details. For Tomcat:
 - Copy this custom `identityiq.war` to a folder under `<Tomcat>/webapps`
 - (e.g. `<Tomcat>/webapps/identityiq`)
 - Navigate to that directory and expand the war: `jar xvf identityiq.war`
 - Delete the war file once you have expanded it

If this is a new deployment (and you don’t need to do a repeatable build – if so, see next heading) or if the build is an upgrade of the IdentityIQ version running on that server, you will need to perform additional actions to create the IdentityIQ database and tables or upgrade the system; consult the IdentityIQ Installation Guide for details as needed. Otherwise you are ready to use your customized IdentityIQ application.

If you wish to update any custom objects and redeploy them to IdentityIQ you can perform the following steps. Open a terminal or command prompt window,

- Navigate to the `<SSB install directory>` folder and enter `build importdynamic`. This command will import all the custom XML artifacts from your `config` folder into IdentityIQ. It will utilize the DB connection from `<environment>.iiq.properties` and import those XML objects into the IdentityIQ database. Target `importdynamic` will not cycle the application server, a step required if changes are made to any class files included in your SSB directory. An alternative to using `build importdynamic` is to manually import the `sp.init-custom.xml` file that was generated during the build, using the command `import sp.init-custom.xml` inside `iiq console`.
- Open IdentityIQ in a web browser and you will see the applications, rules, and other custom objects from your original environment in this new one.
- Note that the SSB modifies the `init.xml` normally used for a “fresh” build of IdentityIQ. The SSB modifies this file from the defaults to import all content (custom objects, LCM objects if desired, and default objects).

Executing a Repeatable, Initial Build of IdentityIQ with SSB

In certain situations (e.g. a non-production environment), it may be appropriate to rebuild an IdentityIQ system many times during development iterations. In a non-production environment, it may be faster to drop the database instead cleaning it up if development tasks go awry – this is especially true during initial deployment phases.

This may mean treating the lowest environment (e.g. dev or sandbox) as “expendable”, which has a real benefit of forcing developers to work from an IDE (Eclipse or IntelliJ, for instance) and keep artifacts in source control. All changes in source control are easily tracked and deployed, while those made in the UI, debug page, or database are harder to measure and port across environments.

Initial Build Prerequisites

To deploy IdentityIQ initially (i.e. install IdentityIQ, create the database, import all objects, etc.), several prerequisites must be met:

- You should have a solid understanding of the dev targets involved and how to target environments properly to control builds – if you do not, proceeding with an “initial build” may put your system at risk and result in data loss. Automation is powerful and can be destructive if used improperly – if there are any doubts, confirm comprehension before executing a build command!
- You should not use an “initial build” for production systems.
- As of SSB v4, MySQL and SQL Server have been tested for “initial build” (Oracle and DB2 may work but have not been as thoroughly tested).
- As of SSB v4, Apache Tomcat has been tested for “initial build” (other application servers may work but have not been tested).
- IdentityIQ 7.0 or higher is recommended for “initial build”. Pre-7.0 versions have not been tested with SSB v4 “initial build”.
- The `createdb` and `dropdb` commands will be used – thus, the `build.properties` settings `db.userid` and `db.password` should have administrative rights to your targeted database instance (create users, delete users, create database, delete database, drop tables, create tables, grant access, etc.). This account will connect to the database server and run SQL scripts. For instance, if SQL Server, use the `sysadmin` server role.
- The database user mentioned by `build.properties` settings `dataSource.username` and `dataSource.password` should not exist when starting an “initial build”, as this may cause the process to fail. These should only be managed via the “initial build” process. This is the user that will be created and managed to connect to the IdentityIQ DB. The user specified here should be also specified as the *IdentityIQ* DB connection user in `iiq.properties` for the build environment.
- If you are using 7.1 or above, there’s a new “plugin” DB, which is separate from the main IdentityIQ DB. The `build.properties` settings of `plugin.db.userName` and `plugin.db.userPassword` should be updated (as this is the account that will access the plugin DB). The user specified here should be also specified as the *plugin* DB connection user in `iiq.properties` for the build environment.
- The `build.properties` settings for the main IdentityIQ DB (`db.name`) and plugin DB (`plugin.db.name` - if 7.1+) should be set as desired.
- You must have properly configured the `build.properties` settings for `db.url` (set as the DB connection string for the DB server where the IdentityIQ DB will be created), `usingLcm`, `console_user`, `console_pass`, `db.type`, and `db.driver`.
- You must be able to place web content where `IIQHome` in `build.properties` points (could be a network share or local path), as using targets like `dist` or `deploy` are mandatory for an “initial build”. Furthermore, this directory should be empty or non-existent.
- The `build.properties` setting `override.safety.prompts` will help avoid safety prompts for `dropdb` and `cleanWeb` (destructive targets).
- If you want to make use of the `cycle` or `up` or `down` targets, you need to have a working script to start or stop your application server (and have the build process running in a security context with rights to do). Furthermore, you need to have the `build.properties` settings `application.server.start` and/or `application.server.stop` configured to point to appropriate scripts.

While the above list may seem daunting, clarifying such items will make the IdentityIQ deployment a refined, smooth process for development iteration.

Initial Build Target Chaining

Installation

Running an “initial build” via the SSB targets will:

- Build the build folder and its extracted contents
- Create an empty, unpatched database
- Apply custom, named object fields to the DB (extend DB schema), if needed
- Import stock (default) IdentityIQ artifacts
- Import stock (default) IdentityIQ LCM artifacts if needed
- Patch the DB if needed
- Run the patch command
- Import custom XML artifacts
- Deploy web content to the web application server

An example “initial build” chained target set might be (this is all 1 line but may appear wrapped):

```
build clean cleanWeb main createdb extenddb import-stock import-lcm patchdb  
runUpgrade import-custom dist
```

You may want to put a flurry of such targets between a `down` and `up` target as well.

The main Ant file `build.xml` has a sample target `initial-build` that encapsulates these targets (no `up` or `down` targets though). Advanced SSB users might tweak this target to provide flexibility in the “initial build” process.

For instance, the above commands would be equivalent to: `build initial-build`

As a bonus, you could cycle the application server: `build down initial-build up`

Removal

Removing IdentityIQ via the SSB targets (reverse of “initial build”) will:

- Stop the application server
- Destroy the IdentityIQ DB (and plugin DB if 7.1+)
- Wipe the web application directory
- Start the application server

Example destroy target set might be: `build main down dropdb cleanWeb clean up`

Note the `main` target here ensures the latest files are placed for `dropdb` to use before the command is executed.

No equivalent wrapper target has been provided for removing IdentityIQ, as the above functions are destructive and should require more deliberate effort to instrument.

Dev targets explained

The following section outlines helpful targets and their application. Note targets can be chained together to be run in sequence and many targets outlined below are simply “helpers” that chain targets. (e.g. `build main dist import-custom cycle` is equivalent to `build importcycle`).

No target (just entering “build” into a windows terminal or “./build.sh” into a Linux terminal)

Runs the entire build process, placing a fully expanded war file in the `<SSB install directory>\build\extract` folder, and all compiled, custom .class files in `<SSB install directory>\build\classes`. This is target “main”.

main

Default Ant target - runs this target when no target is specified.

This target has a hook to run custom Ant scripts - `post.expansion.hook`.

Example: `build` without a target is essentially `build main`.

This target also calls the scripts that perform checks as defined in the Build Checks section of this document.

clean

Deletes everything in the `<SSB install directory>\build` directory.

It is recommended to run the clean target before most deployments, as this ensures a clean working directory.

Examples: `build clean main` or `build clean deploy` or `build clean dist`

cleanWeb

Deletes everything in the directory as specified by build property `IIQHome` (generally the web directory). You should stop the web application server before running this target (either with the `down` target or via another method). It obeys the build property `override.safety.prompts`.

It is recommended to use caution when running this target as it essentially removes the web application. This can be desirable to ensure a clean, fresh deployment directory.

By default, a `build dist` or similar deployment command pushes files to the directory specified by the `IIQHome` build property – it may overwrite files, but does not remove any files. For example, if a file `myJavaLibrary.jar` is in the SSB folder `web/WEB-INF/lib` and deployed to the web server, it will stay on the web server until it is specifically removed. Even if the example jar file is removed from the SSB project folder and a `build clean deploy` is run, `myJavaLibrary.jar` (our example) still remains on

the web server. The `cleanWeb` target is meant to address this issue (specifically in non-production environments).

createdb

Depends on the `build.properties` file having a database account set up that has schema-creation privileges. The properties `db.url`, `db.password`, and `db.userid` in the `build.properties` file must be configured properly to use this build target. This will set up the IdentityIQ schema.

Prior to SSB v4, this target also applied patches to the database. Now, the `patchdb` target handles this action and is called separately.

SSB editions (prior to v4) included an `import-stock` target call, which would attempt to fill the newly-created database with stock and custom artifacts all at once. As of SSB v4, the `import-stock` call has been removed from the `createdb` target. Thus, this target (alone) simply creates a database and tables and leaves them empty.

Note that IdentityIQ 7.1 introduces a new, separate database (`identityiqPlugin` by default). In this release (v4), only the `sqlserver` and `mysql` database types for this target have been updated to work with the new plugin DB. The other types (`oracle` and `db2`) will be addressed in a future release.

cycle

Helper target that depends on both `application.server.start` and `application.server.stop` properties being properly set in `build.properties`. This will cycle your application server and reload the web application. This is equivalent to calling targets `down` and `up` in sequence (with a pause in between). The security context of the build process must have rights to perform the custom script actions.

dropdb

Drops the IdentityIQ database – use with care. Depends on the `build.properties` file having a database account setup that has drop privileges. The properties `db.url`, `db.password`, and `db.userid` in the `build.properties` file must be configured properly to use this build target.

Note that IdentityIQ 7.1 introduces a new, separate database (`identityiqPlugin` by default). In this release (v4), only the `sqlserver` and `mysql` database types for this target have been updated to work with the new plugin DB. The other types (`oracle` and `db2`) will be addressed in a future release.

It obeys the build property `override.safety.prompts`.

When using this target, it is advisable to stop the web application server to close open database connections (otherwise, your drop of the DB may fail). A properly-configured `down` target would suffice here.

dist

Copies the entire expanded war content to your application server `webapps` directory (wherever the `IIQHome` property points to).

dependency-check

Runs the OWASP Dependency Check utility. See the OWASP Dependency Check Vulnerability Detection section of this document for details.

deploy

Runs entire build process and deploys the expanded war content to your application server `webapps` directory (wherever the `IIQHome` property points to) and also import custom XML artifacts. The equivalent of running the build with no target, plus running the `dist` and `import-custom` targets (i.e. `build main dist import-custom == build deploy`).

document

Creates Javadoc-style documentation for the configuration of the XML objects in the `custom` folder. Runs through the main build process first in order to create the objects in that folder, then writes out html files detailing the configuration of each object to the `<SSB install location>/build/doc/TechnicalConfig` folder in the resulting build. Open the `index.html` file in the `TechnicalConfig` folder to view the generated documentation.

The following object types are currently supported by the `document` target:

Application, AuditConfig, Bundle, Capability, CertificationDefinition, CertificationGroup, Configuration, CorrelationConfig, Custom, DashboardContent, Dictionary, Dynamic Scope, EmailTemplate, Form, FullTextIndex, GroupDefinition, GroupFactory, Identity/Workgroup, IdentityTrigger, IntegrationConfig, LocalizedAttribute, ObjectConfig, PasswordPolicy, Policy, QuickLink, QuickLinkOptions, RequestDefinition, Rule, RuleRegistry, ServiceDefinition, SPRight, TargetSource, TaskSchedule, UIConfig, Workflow

When using this feature, you should ensure that application passwords and other sensitive data are not being stored in clear text in the source files or in values substituted by tokens to avoid them being shown in the resulting document.

Note that this documentation is intentionally generated outside of the `build/extract` folder, as it should not be deployed to the web server during a `build deploy` (or similar target action).

down

Runs a custom script, as defined by `application.server.stop` in `build.properties`, to stop the application server. The security context of the build process must have rights to execute your custom script.

extenddb

This target allows for the deployment of *named*, *extended* attributes for objects that support such extensions (Identity, Application, etc.). Prior to SSB v4, field-delivered customizations were needed to deploy DB schema extensions, with the only default object extensions available as numbered columns. This was especially true for initial builds of IdentityIQ using the SSB. SSB v4 delivers this target to meet such needs and allows for low-touch deployments that use named fields.

This target is designed to be used immediately after `createdb`, but it can be run anytime the schema extensions need to be *added*. It will not remove schema extensions, as it acts upon output from `iiq extendedSchema`. This target relies on both the `build.properties` setting `usingDbSchemaExtensions` and having customized Hibernate (`hbm`) and ObjectConfig XML files in the SSB locations `web/WEB-INF/classes/sailpoint/object` and `config/ObjectConfig` (respectively, and the `ObjectConfig` folder is optional). Because there are not special artifact requirements here, this target can be used with or without fresh, initial builds of IdentityIQ.

This target will apply the output from `iiq extendedSchema` to your database, after applying DB naming customizations. Exercise care, as this will actually run a SQL script for your database type (similar to `createdb` and `dropdb` targets).

In SSB v4, only the `sqlserver` and `mysql` database types for this target have been established. The other types (`oracle` and `db2`) will be addressed in a future release.

export

Exports objects specified within `objectsToExport.properties` (this will now be generated – edit `Rule-OutputCustomObjectFile.xml` in `scripts` if you need to add more objects to ignore or export – the variable name of ignored object classes is `listOfIgnoredClasses`) from your `IIQHome` repository to `build/export` so that you don't manually have to copy and paste XML from console-exported files to your build environment. Edit the property file to include all types of objects you want to export, as well as the names of the objects for each type.

import-all

Helper target that simply calls the following targets (in order): `import-stock`, `import-lcm`, `import-custom`.

Attempting to use an import target on a fresh DB (right after `createdb` finishes) will fail if you have named, extended fields for objects (e.g. Identity, Application, etc.). See `extenddb` target for more information.

import-custom

Does a console `iiqBeans` call to `import <build extract location>/WEB-INF/config/sp.init-custom.xml`. This imports all custom XML artifacts into the database. This is usually not called directly and is part of a higher-level call (e.g. `deploy` target). The exception here is a from-scratch build.

Can utilize the `console_user` and `console_pass` build properties.

Attempting to use an import target on a fresh DB (right after `createdb` finishes) will fail if you have named, extended fields for objects (e.g. Identity, Application, etc.). See `extendddb` target for more information.

import-lcm

Does a `console iiqBeans call to import <build extract location>/WEB-INF/config/init-lcm.xml`. This imports all default Lifecycle Manager (LCM) XML artifacts into the database. This target is generally used for a from-scratch build.

Can utilize the `console_user` and `console_pass` build properties.

Attempting to use an import target on a fresh DB (right after `createdb` finishes) will fail if you have named, extended fields for objects (e.g. Identity, Application, etc.). See `extendddb` target for more information.

import-stock

Does a `console iiqBeans call to import <build extract location>/WEB-INF/config/init-default_org.xml`. This imports the stock (i.e. out-of-the-box) XML items that represent a base artifact set for IdentityIQ. Note that the SSB renames the default `init.xml` to be `init-default_org.xml`. This means importing `init.xml` from the build extract area includes customized elements (i.e. like `sp.init-custom.xml`). Thus, the `import-stock` target merely imports default XML artifacts (it will not import your customizations). This target is generally used for from-scratch builds. This target does not import LCM-related elements.

Can utilize the `console_user` and `console_pass` build properties.

Attempting to use an import target on a fresh DB (right after `createdb` finishes) will fail if you have named, extended fields for objects (e.g. Identity, Application, etc.). See `extendddb` target for more information.

import (deprecated)

Deprecated target as of SSB v4. Maintained only for backwards compatibility at this time. Duplicates the functionality of target `import-custom`. New work using the SSB should not use this target; use `import-custom` instead.

importcycle

Helper target that runs the entire build process, imports custom XML artifacts, copies Java classes and static web content, and cycles (restarts) the application server. Useful while developing custom Java. (i.e. equivalent to `build main dist import-custom cycle`).

Can utilize the `console_user` and `console_pass` build properties (indirectly).

importdynamic

Helper target that runs the entire build process and imports some content that does not require an application reload: custom XML, static web content etc. Useful for developing rules, workflow and branding (i.e. equivalent to `build main dist import-custom`).

Can utilize the `console_user` and `console_pass` build properties (indirectly).

importjava

Helper target that runs the entire build process, copies Java classes and static web content, and cycles (restarts) the application server. Useful while developing custom Java and you want to skip importing custom XML artifacts. (i.e. equivalent to `build main dist cycle`).

initial-build

Helper target that runs several other targets. Requires careful planning but allows for rapid, repeatable builds of an IdentityIQ environment.

Note: `initial-build` target is not recommended for production environments.

See section Executing a Repeatable, Initial Build of IdentityIQ with SSB for more details.

patchdb

Prior to SSB v4, a call to `createdb` would implicitly run the `patch sql` script (`upgrade_identityiq_tables... version` and DB-specific). Now, that functionality has been broken out into the target `patchdb`. This target is only effective if there is a patch level for IdentityIQ specified in `build.properties`.

runSql

Can run arbitrary SQL scripts of your choosing against the IdentityIQ DB. This leverages the same `build.properties` settings as other DB-related targets (i.e. `createdb`). Also like the other DB-related targets, this uses the Ant `sql` task.

This target differs from most others in the SSB, as target properties can be passed in during the call. A `-D` prefix is used with each property (e.g. `-Dproperty1=value`).

The table below shows the valid properties for command line use. Technically, these could also be defined in `build.properties`. They are not defined there by default to increase the flexibility of the target call.

Name	Description	Required
<code>sql.input.file</code>	SQL script file you want to invoke against the IdentityIQ DB	Yes

sql.output.file	File to receive query output. Can be useful if query output should be saved or used for other automation.	No. If omitted, output is only printed to the screen (standard out).
sql.error.action	Specifies if a SQL error should stop script processing. Sometimes, there are errors that can be safely ignored.	No. If omitted, action is “abort” on any SQL script error.

In addition to passing in properties for this command, know that Ant properties in the SQL script will be expanded. Thus, instead of hard-coding your DB name for the script (as it will be the IdentityIQ DB) like:

```
create table identityiq.myTable;
```

You could use the statement below because your `build.properties` settings will be expanded when the SQL script runs:

```
create table ${db.name}.myTable;
```

You could really define any arbitrary property for SQL script expansion that you’d like (either in the relevant `build.properties` file (recommended) or ad-hoc via the command line.

Below are a few use case examples.

Example command (Windows pathing) that runs some SQL and saves output. Continues on error.

```
build runSql -Dsql.input.file=C:\input.sql -Dsql.output.file=C:\output.sql -Dsql.error.action=continue
```

Example command (Windows pathing) that runs some SQL and prints output to screen. Aborts on error.

```
build runSql -Dsql.input.file=C:\input.sql
```

If `sql.input.file` is defined in `build.properties`, you could run the command without passing in command-line parameters. Note that this reduces flexibility of the command if multiple files need to be processed.

```
build runSql
```

runUpgrade

Runs the `patch` command via `sailpoint.launch.Launcher`, which applies a patch to an IdentityIQ installation. This is generally only needed when creating a new install via “initial build”.

Can utilize the `console_user` and `console_pass` build properties.

up

Runs a custom script, as defined by `application.server.start` in `build.properties`, to start the application server. The security context of the build process must have rights to execute your custom script.

war

Make a war file from the output generated by target `main` in the `build/extract` directory. A war file is essentially a compressed archive file (known as **Web Application Archive**) that can be used for deploying web applications to Tomcat or similar servers. The generated war file, `identityiq.war`, is put into `build/deploy`.

This target includes a hook to run custom Ant scripts - `post.war.hook`.

Build Checks

SSB v4 adds a series of checks around the build process to alert the deployer to common misconfigurations and deviations from best practices. The results of these checks are written to a log file.

Please note these checks are only indicative in nature and can result in false positives.

Controlling Build Checks

Build checks are disabled by default. They can be enabled by setting the property `runCodeChecks` to `true` in the `build.properties` file.

Available Build Checks

There are two sets of build checks, one which is executed after the expansion phase and another which takes place after token substitution.

Checks after SSB Build Expansion Phase

The following checks are performed after the base files used for the build are expanded.

1. **Wildcard import check.** XML files are checked to verify whether they have any wildcard import statements in BeanShell code. Best practice is to avoid this because it is associated with a performance overhead.

For example: `import sailpoint.object.*;`
2. **Code comments check.** Java files are checked for the absence of code comments. Best practice is to add comments that will make your code easy for another developer to understand.
3. **System.out check.** Java files are checked for the presence of `System.out` statements. Best practice is to use log statements which can be switched on or off using Log4j log levels.
4. **Naming convention check.** All XML files are checked to verify whether they adhere to a specified naming convention.

The naming convention is specified with the `codeCheck.namingConvention` property in `build.properties`. This property takes a Java Regular Expression and recognizes two keywords: `OBJECT` and `FOLDER`. `OBJECT` represents the object class represented in an XML file (e.g. Rule or Workflow). `FOLDER` represents the subfolder directly under the SSB `config` folder where an XML file is located.

For example, if you are working on a project called “XYZ” where the naming convention specifies files in this format:

```
<project name>-<object type>-<free text description>.xml
```

you could use this as the naming convention in `build.properties`:

```
codeCheck.namingConvention=XYZ-OBJECT-.*.xml
```

This would match the file `XYZ-Application-ActiveDirectory.xml` if that file contains an IdentityIQ Application object. The file could be anywhere in the directory structure under the `config` folder.

Alternatively, if the naming convention specifies this format:

```
<project name>-<folder name>-<free text description>.xml
```

you could use this in `build.properties`:

```
codeCheck.namingConvention=XYZ-FOLDER-.*.xml
```

This will match the file `XYZ-Rule-AD_Correlation.xml` but only if that file is located directly under the `config/Rule` folder.

It is possible to exclude specific folders from the naming convention check by editing the `ignoreFolderList` property in the `verifyNamingConvention` entry in the file `scripts/build.check.xmlVerifyNamingConvention.xml`. This contains a comma-separated list of folders to ignore. By default it will exclude the `SSF_Tools`, `SSF_Features`, `SSF_Frameworks` and `SSP_Tools` folders from the check:

```
ignoreFolderList="SSF_Tools,SSF_Features,SSF_Frameworks,SSP_Tools"
```

When the `OBJECT` keyword is specified and more than one object type is represented in an individual file, the task will exclude that file from the naming convention check.

Any file or folder which is excluded from the check will be written to the log under the section “Skipped for Name Convention checks”.

Checks after SSB Token Substitution Phase

The following checks are performed after tokens have been substituted from the `target.properties` files.

1. **Application password encryption check.** All XML files that represent Application objects are checked to verify whether they have passwords that are not encrypted.

2. **iiq.properties password encryption check.** All `iiq.properties` files are checked to verify whether they have passwords that are not encrypted.
3. **Workflow trace enabled check.** All Workflow XML objects are checked to verify whether the `trace` variable has been initialized to `true`. This should normally be set to `false` in production environments to avoid unnecessary logging which could affect performance.
4. **BeanShell code deprecation check.** BeanShell code in the build is scanned to determine whether there are any methods that are deprecated in the version of IdentityIQ that the build is running for. This can be particularly useful when you are upgrading IdentityIQ, as manually checking your BeanShell code for deprecations can be a time-consuming process. In the current version of the deprecation scanner code there are a few limitations to be aware of:
 - Classes declared within BeanShell are not handled, and invocations within can result in false positives
 - Any catch/finally blocks are not handled, and invocations within them are not scanned
 - Chained invocations like `method1().method2().method3()` are not processed
 - If a deprecated method is invoked multiple times it will be listed multiple times in the log
 - Variables representing objects where the object type is not declared can result in false positives. When the scanner cannot determine the object type it checks only the method portion against all deprecations known within the BeanShell namespace and indicates any found as a “possible deprecated method” in the log.
 - Missing import statements can also result in “possible deprecation” in the log.

Project-Specific Build Checks

An implementation team can choose to add their own build Checks to enforce or validate project-specific rules by creating new build check scripts, using the provided scripts as examples. All build checks are implemented as Ant script files and reside within the `scripts` folder. The following naming convention for build check scripts that are performed after the SSB expansion phase is:

```
build.check.<name>.xml
```

The naming convention for build check scripts that are performed after the SSB token substitution phase is:

```
build.postcheck.<name>.xml
```

Each build check script is required to implement a `sp.services.runCodeChecks` target.

Build Check Output

Once the build checks are complete, the output is generated in the file

```
build/buildChecks/buildCheck.log.
```


OWASP Dependency Check Vulnerability Detection

The Dependency Check Utility

The Open Web Application Security Project (OWASP) provides a utility called Dependency Check, which identifies project dependencies and checks if there are any known, publicly disclosed, vulnerabilities. The Ant implementation of this is included in the SSB and can check for vulnerabilities in any third-party libraries that ship with IdentityIQ or are added by implementers. An HTML report is generated, including the Common Vulnerabilities and Exposures (CVE) number, a description, a severity classification and number, and the affected library. Clicking on the CVE provides further details on the vulnerability.

Part of a generated Vulnerability Report for IdentityIQ is shown below. The vulnerabilities listed here are related to the MySQL and SQL Server driver files. These are showing on the report because the versions of these files that ship with the product are not the most recent (but provided to ensure backwards compatibility) and are subject to some known vulnerabilities; this is one reason why it is always recommended to update the JDBC drivers (see the JDBC Drivers section of this document under Build Structure Set-up).



Dependency-Check is an open source tool performing a best effort analysis of 3rd party dependencies; false positives and false negatives may exist in the analysis performed by the tool. Use of the tool and the reporting provided constitutes acceptance for use in an AS IS condition, and there are NO warranties, implied or otherwise, with regard to the analysis or its use. Any use of the tool and the reporting provided is at the user's risk. In no event shall the copyright holder or OWASP be held liable for any damages whatsoever arising out of or in connection with the use of this tool, the analysis performed, or the resulting report.

About The Vulnerability Report | Getting Help: [google group](#) | [github issues](#)

This report is intended to be a quick summary of findings. It is highly recommended that you use the full HTML report to determine if any [false positives](#) have been reported. Additionally, the HTML report provides many features not found in the vulnerability report.

Vulnerability Report for IdentityIQ

Report Generated On: Jul 25, 2018 at 14:49:46 +01:00

Dependencies Scanned: 218
Vulnerable Dependencies: 23

Vulnerable Dependencies

NAME	CWE	Severity (CVSS):	Dependency
CVE-2016-0639		High(10.0)	mysql-connector-java-5.1.28-bin.jar
CVE-2016-0705		High(10.0)	mysql-connector-java-5.1.28-bin.jar
CVE-2016-6662	CWE-264 Permissions, Privileges, and Access Controls	High(10.0)	mysql-connector-java-5.1.28-bin.jar
CVE-2012-1856	CWE-94 Improper Control of Generation of Code ("Code Injection")	High(9.3)	sqljdbc4.jar
CVE-2012-3163		High(9.0)	mysql-connector-java-5.1.28-bin.jar
CVE-2015-1763	CWE-284 Improper Access Control	High(8.5)	sqljdbc4.jar
CVE-2014-6507		High(8.0)	mysql-connector-java-5.1.28-bin.jar
CVE-2017-3599	CWE-284 Improper Access Control	High(7.8)	mysql-connector-java-5.1.28-bin.jar

Running the Dependency Check Utility

To run the Dependency Check utility in the SSB, run the dependency-check target (`build dependency-check`). This should be done from a computer that has Internet access as it needs to download and process the latest data from the National Vulnerability Database (NVD) hosted by NIST: <https://nvd.nist.gov>.

Running this target will first expand the product files and patches so that the library files are ready for analysis. The first time the utility is executed it will download the latest vulnerability data. It will then analyze the library files in the `build/extract/WEB-INF/lib` folder to determine whether there are any matching vulnerabilities. Reports will then be generated in the `build/dependency-check-reports` folder.

A detailed Dependency Check report is generated in HTML, CSV, XML and JSON formats. A summary Dependency Check Vulnerability report (as shown above) is generated in HTML format.

The suppressions.xml File

The Dependency Checker includes a suppressions.xml file in the dependency-check-ant folder at the root of the SSB, which suppresses reporting on CVEs for libraries that are included in IdentityIQ where the product is known to be unaffected by the vulnerability. This is in the format of a series of <suppress> entries like this:

```
<suppress>
  <notes><![CDATA[
    This suppresses CVE-2017-5662 for batik-awt-util-1.6-1.jar.
    The documented CVE for this version of this library is related to an XML Entity Attack while parsing SVG files.
    IdentityIQ does not contain any support for SVG formatted files or parsing of them and therefore is not vulnerable.
  ]]></notes>
  <filePath regex="true">.*\bbatik-awt-util-1\..6-1\.jar</filePath>
  <cve>CVE-2017-5662</cve>
</suppress>
```

This includes information about the library, the CVE and an explanation of why the detected vulnerability does not affect IdentityIQ. In time, new vulnerabilities may be found, and if IdentityIQ is not affected by these vulnerabilities the suppressions.xml file may need to be updated to suppress reporting of them. SailPoint hopes to be able to provide and maintain a list of these detected vulnerabilities on Compass in the near future so that questions resulting from any vulnerabilities exposed by this utility can be quickly addressed.

The suppressions.xml file also contains a <suppress> entry for excluding any CVEs that have a score below a given value. By default this value is 7, which represents the number above which a CVE is given a severity of 'High'.

```
<suppress>
  <notes><![CDATA[
    This suppresses all CVE entries that have a score below CVSS 7.
  ]]></notes>
  <cvssBelow>7</cvssBelow>
</suppress>
```

Updating the Dependency Check Utility

At times you may see a message like this in the output screen when you run the dependency-check target if there is a new version available:

```
[dependency-check] A new version of dependency-check is available. Consider updating to version 3.2.1.
```

The latest version can be downloaded from this location:

<https://jeremylong.github.io/DependencyCheck/dependency-check-ant/index.html>

This will be in the form of a zip file that can be expanded to replace the existing dependency-check-ant folder at the root of the SSB. Ensure you copy across the existing suppressions.xml file.

Note that the latest version at the time of writing (July 2018) is 3.3.0. During testing, SailPoint has found issues with this version which prevent the check from completing. For that reason, the previous

version (3.2.1) is the version that currently ships with the SSB. SailPoint advises customers and partners to wait for the next version after 3.3.0 before updating.

Further Information on the Dependency Check Utility

Further information on the OWASP Dependency Check utility can be found here:

https://www.owasp.org/index.php/OWASP_Dependency_Check

Information on the Ant implementation can be found here:

<https://jeremylong.github.io/DependencyCheck/dependency-check-ant/index.html>