Reference Guide

Reference Guide: Open Build Service

by Adrian Schröter, Frank Schreiner, Karsten Keil, and Stefan Knorr

Publication Date: 01/23/2019

SUSE LLC
10 Canal Park Drive
Suite 200
Cambridge MA 02141
USA
https://www.suse.com/documentation

Copyright © 2006– 2019 SUSE LLC and contributors. All rights reserved.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or (at your option) version 1.3; with the Invariant Section being this copyright notice and license. A copy of the license version 1.2 is included in the section entitled “GNU Free Documentation License”.

For SUSE trademarks, see http://www.suse.com/company/legal/. All other third-party trademarks are the property of their respective owners. Trademark symbols (®, ™ etc.) denote trademarks of SUSE and its affiliates. Asterisks (*) denote third-party trademarks.

All information found in this book has been compiled with utmost attention to detail. However, this does not guarantee complete accuracy. Neither SUSE LLC, its affiliates, the authors nor the translators shall be held liable for possible errors or the consequences thereof.

Contents

About this Guide viii

1 OBS Architecture 1

1.1 Overview Graph 1

1.2 Communication Flow 4

2 OBS Concepts 7

2.1 Project Organization 7

Project Metadata 7 • Project Build Configuration 8 • Project Build Macro Configuration 9 • An OBS Package 9

2.2 The OBS Interconnect 9

2.3 Download on Demand Repositories (DoD) 9

Motivation 9 • XML Document Hierarchy 10 • The Daemon 10 • The `download` Element 10 • The `master` Subelement 11 • The `pubkey` Subelement 11 • Repository Types 11

2.4 Integrating External Source Repositories 14

Motivation 14 • Creating an Reference to an External SCM 15 • Working with Local Checkouts 16 • Managing Build Recipes in a SCM 16

3 Build Process 17

3.1 Phases of a Build Process 17

Preinstall Phase 17 • Install Phase 17 • Package Build 18 • After the Build 18

3.2 Identify a build 18

Read DISTURL from a rpm 18 • Read DISTURL from a container 19

4 Source Management 20

4.1 Find Package Sources 20

5 Request And Review System 21

5.1 What a request looks like 21

Action Types 22 • Request states 23 • Reviewers 24 • Request creation 24 • Request operations 24

6 Image Templates 26

6.1 Structure of Image Templates 26

6.2 Adding Image Templates to/Removing Image Templates from the Official Image Template Page 26

6.3 Receiving Image Templates via Interconnect 26

7 Multiple Build Description File Handling 28

7.1 Overview 28

7.2 How Multibuild is Defined 28

8 Maintenance Support 30

8.1 Simple Project Setup 31

8.2 Using the Maintenance Process 32

Workflow A: A Maintainer Builds an Entire Update Incident for Submission 32 • Workflow B: Submitting a Package Without Branching 33 • Workflow C: Process Gets Initiated By the Maintenance Team 34 • Maintenance Incident Processing 34 • Incident Gets Released 35 • Incident Gets Reopened and Re-Released 35 • Using Custom Update IDs 36

8.3 OBS Internal Mechanisms 36

Maintenance Incident Workflow 36 • Searching for Incidents 38

8.4 Setting Up Projects for a Maintenance Cycle 39

Defining a Maintenance Space 39 • Maintained Project Setups 39

8.5 Optional Channel Setup 39

Defining a Channel 39 • Using Channels in Maintenance Workflow 40

9 Binary Package Tracking 43

9.1 Which Binaries Are Tracked? 43

9.2 What Data Is Tracked? 43

Binary Identifier 43 • Binary Information 44 • Product information 44

9.3 API Search Interface 44

Examples 45

10 Administration 46

10.1 OBS Components 46

Front-end 48 • Back-end 48 • Command Line Client 50 • Content Delivery Server 50 • Requirements 50

10.2 OBS Appliances 51

Server Appliance 51 • Worker Appliance 51 • Image Types 51 • Deployment 52 • Separating Data from the System 52 • Updating the Appliance 53

10.3 Back-end Administration 53

Services 53 • Advanced Setups 54

10.4 Front-end Administration 54

Delayed Jobs 55 • Full Text Search 55

11 Scheduling and Dispatching 56

11.1 Definition of a Build Process 56

11.2 Scheduling Strategies 56

Build Trigger Setting 57 • Block Mode 57 • Follow Project Links 58

12 Build Job Constraints 59

12.1 `hostlabel` 59

12.2 `sandbox` 60

12.3 `linux` 60

version 60 • `flavor` 61

12.4 `hardware` 61

`cpu` 61 • `processors` 62 • `disk` 62 • `memory` 63 • `physicalmemory` 63

12.5 Constraint Handling 64

More than half of the workers satisfy the constraints 64 • Less than Half of the Workers Satisfy the Constraints 64 • No Workers Satisfy the Constraints 64

12.6 Checking Constraints with **`osc`** 65

13 Building Preinstall Images 67

14 Authorization 68

14.1 OBS Authorization Methods 68

Default Mode 68 • Proxy Mode 68 • LDAP Mode 68

14.2 OBS Token Authorization 68

Managing User Tokens 69 • Executing a Source Service 69

15 Quality Assurance(QA) Hooks 71

15.1 Source Related Checks 71

15.2 Build Time Checks 71

In-Package Checks 71 • Post Build Checks 72 • Post Build Root Checks 72 • KIWI Specific Post Build Root Checks 72

15.3 Workflow Checks 72

Automated Test Cases 72

16 openSUSE Factory 74

16.1 openSUSE:Factory project 74

16.2 Devel Projects 74

Glossary 75

A GNU Licenses 85

A.1 GNU General Public License 85

A.2 GNU Free Documentation License 87

About this Guide

This guide is part of the Open Build Service documentation. These books are considered to contain only reviewed content, establishing the reference documentation of OBS.

This guide does not focus on a specific OBS version. It is also not a replacement of the documentation inside of the openSUSE Wiki (https://en.opensuse.org/Portal:Build_Service). However, content from the wiki may be included in these books in a consolidated form.

1 Available Documentation

The following documentation is available for OBS:

Book "Administrator Guide"

This guide offers information about the initial setup and maintenance for running Open Build Service instances.

Article "Beginner's Guide"

This guide describes basic workflows for working with packages on Open Build Service. This includes checking out a package from an upstream project, creating patches, branching a repository, and more.

Book "Best Practice Guide"

This guide offers step-by-step instructions for the most common features of the Open Build Service and the openSUSE Build Service.

Reference Guide

This guide covers ideas and motivations, concepts and processes of the Open Build Service and also covers administration topics.

Book "User Guide"

This guide is intended for users and developers who want to dig deeper into Open Build Service. It contains information on backgrounds, setting up your computer for working with OBS, and usage scenarios.

2 Feedback

Several feedback channels are available:

Bugs and Enhancement Requests

Help for openSUSE is provided by the community. Refer to https://en.opensuse.org/Portal:Support for more information.

Bug Reports

To report bugs for Open Build Service, go to https://bugzilla.opensuse.org/, log in, and click *New*.

Mail

For feedback on the documentation of this product, you can also send a mail to `doc-team@suse.com`. Make sure to include the document title, the product version and the publication date of the documentation. To report errors or suggest enhancements, provide a concise description of the problem and refer to the respective section number and page (or URL).

3 Documentation Conventions

The following notices and typographical conventions are used in this documentation:

- `/etc/passwd`: directory names and file names
- *`PLACEHOLDER`*: replace *`PLACEHOLDER`* with the actual value
- `PATH`: the environment variable PATH
- **`ls`**, `--help`: commands, options, and parameters
- `user`: users or groups
- `package name` : name of a package
- `Alt`, `Alt`–`F1`: a key to press or a key combination; keys are shown in uppercase as on a keyboard
- *File*, *File* › *Save As*: menu items, buttons
- *Dancing Penguins* (Chapter *Penguins*, ↑Another Manual): This is a reference to a chapter in another manual.

- Commands that must be run with `root` privileges. Often you can also prefix these commands with the **`sudo`** command to run them as non-privileged user.

```
root # command
geeko > sudo command
```

- Commands that can be run by non-privileged users.

```
geeko > command
```

- Notices

Warning: Warning Notice

Vital information you must be aware of before proceeding. Warns you about security issues, potential loss of data, damage to hardware, or physical hazards.

Important: Important Notice

Important information you should be aware of before proceeding.

Note: Note Notice

Additional information, for example about differences in software versions.

Tip: Tip Notice

Helpful information, like a guideline or a piece of practical advice.

4 Contributing to the Documentation

The OBS documentation is written by the community. And you can help too!

Especially as an advanced user or an administrator of OBS, there will be many topics where you can pitch in even if your English is not the most polished. Conversely, if you are not very experienced with OBS but your English is good: We rely on community editors to improve the language.

This guide is written in DocBook XML which can be converted to HTML or PDF documentation.

To clone the source of this guide, use Git:

```
git clone https://github.com/openSUSE/obs-docu.git
```

To learn how to validate and generate the OBS documentation, see the file `README`.

To submit changes, use GitHub pull requests:

1. Fork your own copy of the repository.
2. Commit your changes into the forked repository.
3. Create a pull request. This can be done at https://github.com/openSUSE/obs-docu.

It is even possible to host instance-specific content in the official Git repository, but it needs to be tagged correctly. For example, parts of this documentation are tagged as `<para os="opensuse">`. In this case, the paragraph will only become visible when creating the openSUSE version of a guide.

1 OBS Architecture

1.1 Overview Graph

Open Build Service (OBS) is not a monolithic server; it consists of multiple daemons that fulfill different tasks (see *Figure 1.1, "Simplified OBS Component Overview"*).

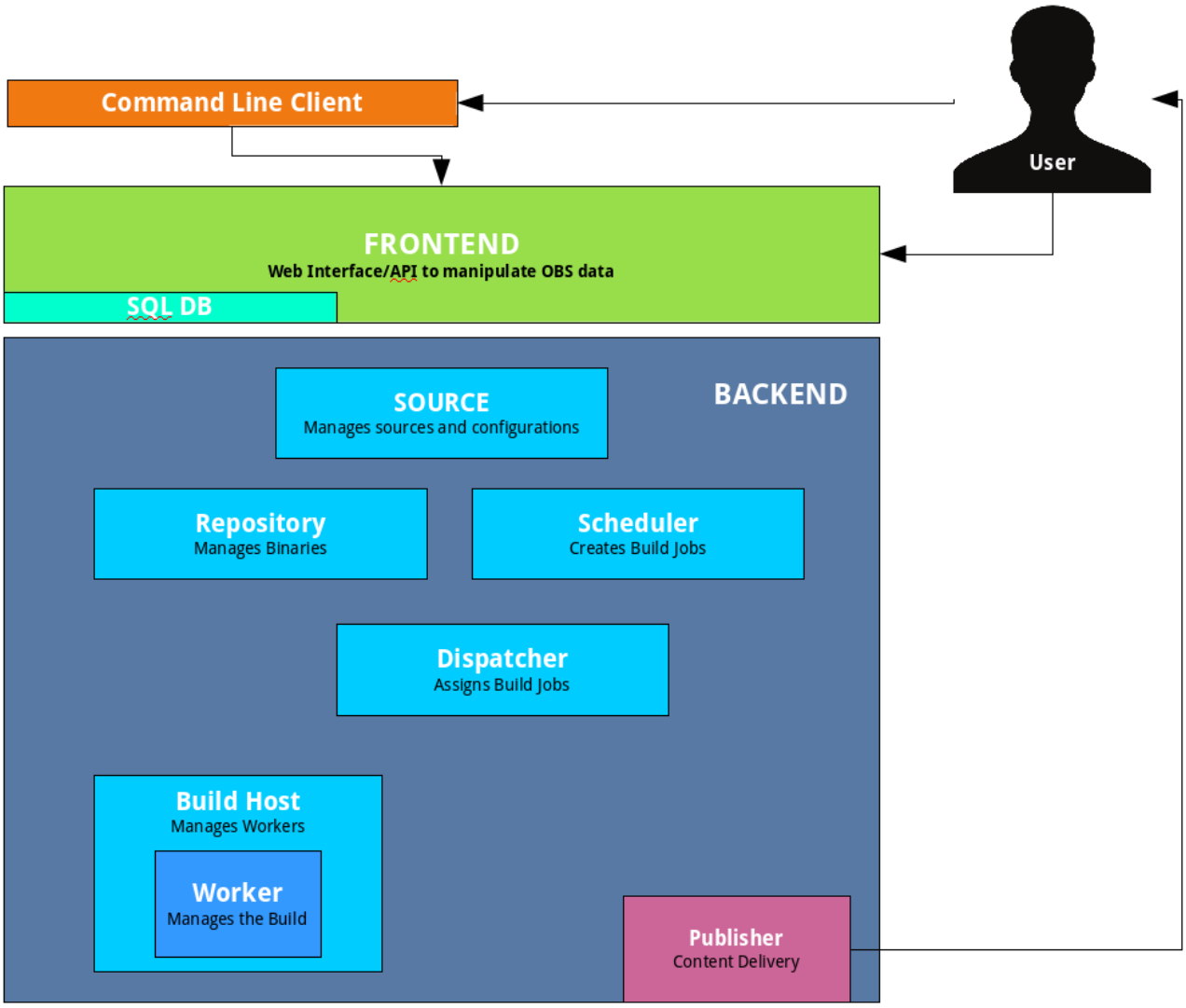


FIGURE 1.1: SIMPLIFIED OBS COMPONENT OVERVIEW

The OBS Back-end manages the source files and build jobs of the OBS.

Source Server

Maintains the source repository and project/package configurations. It provides an HTTP interface, which is the only interface for the Front-end and It may forward requests to further back-end services.

The Source Server keeps track of all sources that are available for building. It takes care of file deduplication so that every source file is stored only once. This is done by keeping track of the MD5 hashes of the files in combination with the file names. All revisions of committed sources are stored and will not be deleted. This guarantees the ability to get the source for every delivered binary package.

Each OBS installation has one Source Server only. It maintains the "sources", "trees" and "projects" directories.

Repository Server

A repository server provides access to the binaries via an HTTP interface. It is used by the front-end via the source server only. Workers use the server for registration, requesting the needed binaries for the build jobs and storing the result. Notifications for schedulers are also created by repository servers. Each OBS installation has at least one repository server. A larger installation using partitioning has one on each partition.

Scheduler

A scheduler calculates the need for build jobs. It detects changes in sources, project configurations or in binaries used in the build environment. It is responsible for starting jobs in the right order and integrating the built binary packages. Each OBS installation has one scheduler per available architecture and partition. It maintains the cntent of the "build" directory.

Dispatcher

The dispatcher takes a job (created by the scheduler) and assigns it to a free worker. It also checks possible build constraints to verify that the worker qualifies for the job. It only notifies a worker about a job; the worker downloads the needed resources itself afterwards. Each OBS installation has one dispatcher per partition, one of which is the master dispatcher.

The dispatcher tries to assign jobs fairly between the project repositories. For this the dispatcher maintains a **load** per project repository (similar to the Unix system load) of used build time. The dispatcher assigned jobs to build clients from the repository with the lowest load (thereby increasing its load). It is possible to tweak this mechanism via

dispatching priorities assigned to the repositories via the */build/*dispatchprios ***API*** call or via the **dispatch_adjust** map in the BSConfig.pm configuration file. See the dispatch priorities in reference guide for more details.

Publisher

The publisher processes publish events from the scheduler for finished repositories. It merges the build result of all architectures into a defined directory structure, creates the needed metadata, and may sync it to a download server. It maintains the content of the "repos" directory on the back-end. Each OBS installation has one publisher per partition.

Signer

The signer handles signing events and calls an external tool to execute the signing. Each OBS installation usually has one signer per partition and also on the source server installation.

Source Service Server

The Source Service Server helps to automate processes for **continuous integration**. The server can call different services for different tasks. It can download sources from websites and version control systems such as subversion and git. Services can also include working on the source to extract spec-files from archives, repacking the archives or adjusting version numbers in spec files. It is also often used to enforce policies by running checks. A failed check will appear as broken source and blocks a package from building.

The Source Service Server is optional and currently only one Source Service Server is supported.

Download on Demand Updater (dodup) (OBS version 2.7 or later)

The download on demand updater monitors all external repositories which are defined as download on demand resources. It polls for changes in the metadata and re-downloads the metadata in case. The scheduler will be notified to recalculate the build jobs depending on these repositories afterwards. Each OBS installation can have one dodup service running on each partition.

Delta Store (OBS version 2.7 or later)

The delta store daemon maintains the deltas in the source storage. Multiple obscpio archives can be stored in one deltastore to avoid duplication on disk. This service calculates the delta and maintains the delta store. Each OBS installation can have one delta store process running next to the source server.

Worker

The workers register with the repository servers. They receive build jobs from the dispatcher. Afterwards they download sources from the source server and the needed binaries from the repository server(s). They build the package using the build script and send the result back to the repository server. A worker can run on the same host as the other services, but most OBS installations have dedicated hardware for the workers.

1.2 Communication Flow

The communication flow can be split into 3 parts:

1. User-to-front-end communication
2. Front-end-to-source-server communication
3. communication between source server and other back-end components, in first place the repository servers.

The user is using the front-end (via tools like ***osc***) to communicate with the Open Build Service. The front-end is providing a web interface and also an API. The front-end is implemented as Ruby on Rails application. All communication happens via the ***http*** protocol (normally encrypted so ***https*** is used.

The communication between the front-end and the back-end also use the http protocol, using the back-end source server as gateway to most of the other back-end components.

The figure *Figure 1.2, "OBS Communication"* shows the communication flow between the OBS components if a package source (e.g. a _service file) was updated.

If a source file of a package was updated, the new source file is uploaded with an HTTP PUT operation to the front-end. The front-end may ask for authentication (if not already done) and check the access rights for the user. If everything is OK, the new file will be send to the back-end ***Source server*** via an HTTP PUT request as well. The source server stores the changed source (under revision control) and check if for this change source service runs are needed. If yes the ***Source Service server*** is informed via an HTTP PUT request of the _service file to run the requested services.

The ***Source Service server*** runs the different source services, e.g download the requested revision from a GIT server. After running all services it delivers the final sources back to the ***Source server***, which then store these under revision control as well.

The ***Source server*** then notifies the ***Schedulers*** (for each architecture) about the change of the package via an event.

The ***Scheduler*** recalculates the package and project state and if all build requires are solved, a build Job is created and put in the ***Job pool***. The notified ***Dispatcher*** checks for free worker which met the build constraints for the job and send the job to the ***worker***.

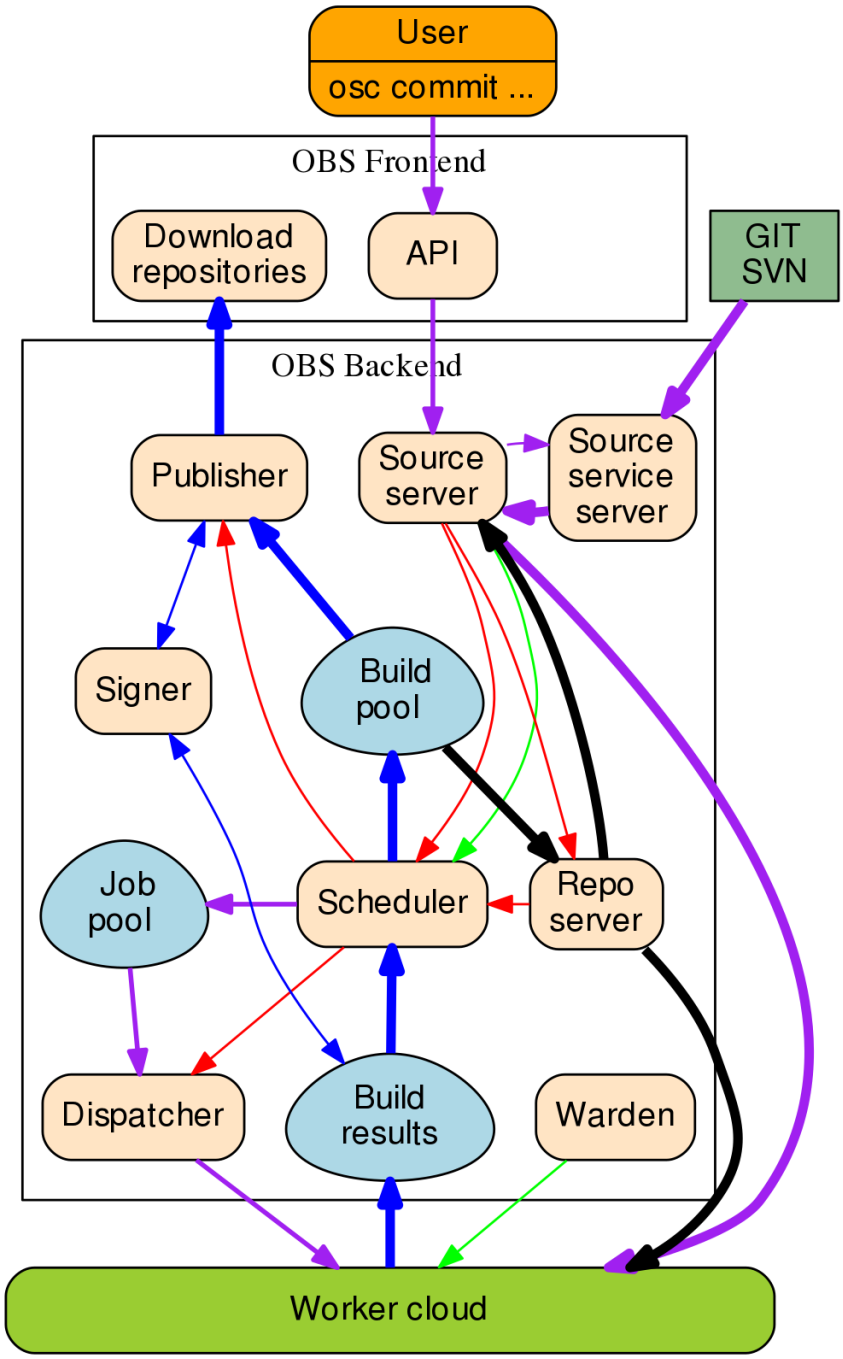


FIGURE 1.2: OBS COMMUNICATION

The ***Worker*** downloads the sources from the ***Source server*** and all required binary packages from the ***Repo server***. The package then will be built, the worker is monitored by the ***warden*** service to detect any worker crashes.

If the build was finished the build results (which are also contain build logs) the results are uploaded to the ***Scheduler*** again and if requested the ***Signer*** does sign the packages. The ***Scheduler*** recalculate the project status if here are more dependent packages to build. If not it informs the ***Publisher*** to publish the new build results and create an updated version of the repository. The ***Publisher*** will also request signing of the repository metadata.

2 OBS Concepts

We describe here the high-level concepts: how Open Build Service is designed, manages its content and is supposed to work.

2.1 Project Organization

All sources and binaries which are hosted inside of OBS are organized into projects. A project is the container defining a larger task. It defines who is working there.

2.1.1 Project Metadata

A project is configured in the project **`/source/$PROJECT/_meta`** path. It can be edited in the web interface using the **`RAW Config`** tab or via command line with

```
osc meta prj -e $PROJECT
```

This file contains:

- Generic description data in **`title`** and **`description`** elements.
- An ACL list of users and groups connected with a role. The **`maintainer`** role defines the list of users permitted to commit changes to the project.
- A number of flags controlling the build and publishing process and possible read access protections.
- A list of repositories to be created. This list defines what other repositories should be used, which architectures shall be built and build job scheduling parameters.

The following flags can be used to control the behavior of a package or project. Most of them can also be limited to specified repositories or architectures.

- **`build`** defines whether package sources should get built. If enabled, it signals the scheduler to trigger server-side builds based on events like source changes, changes of packages used in the build environment or manual rebuild triggers. A local build via CLI is possible independent of this flag. Default is enabled.
- **`publish`** can be used to enable or disable publishing the build result as repository. This happens after an entire repository has finished building for an architecture. A publish also gets triggered when the publish flag is enabled after a repository finishes the build. Default is enabled.
- **`debuginfo`** can be used to modify the build process to create debuginfo data along with the package build for later debugging purposes. Changing this flag does not trigger rebuilds, it just affects the next build. Default is disabled.
- **`useforbuild`** is used to control if a built result shall be copied to the build pool. This means it will get used for other builds in their build environment. When this is disabled, the build has no influence on builds of other packages using this repository. In case a former build exists the old binaries will be used. Disabling this flag also means that "wipe" commands to remove binary files will have no effect on the build pool. Changing this flag does not trigger rebuilds, it just affects the next build. Default is enabled.
- **`access`** flag can be used to hide an entire project. This includes binaries and sources. It can only be used at project creation time and can just be enabled (making it public again) afterwards. This flag can only be used on projects. Default is enabled.
- **`sourceaccess`** flag can be used to hide the sources, but still show the existence of a project or package. This also includes debug packages in case the distribution is supporting this correctly. This flag can only be used at package creation time. There is no code yet which checks for possible references to this package. Default is enabled.
- **`downloadbinary`** permission still exists like before. However, unlike "access" and "sourceaccess" this is not a security feature. It is just a convenience feature, which makes it impossible to get the binaries via the API directly. But it is still possible to get the binaries via build time in any case. Default is enabled.

2.1.2 Project Build Configuration

A project is configured in the project **`/source/$PROJECT/_config`** path. It can be edited in web interface in the **`Project Config`** tab or via one of the following command lines

```
osc meta prjconf -e $PROJECT
osc co $PROJECT _project
```

This file contains information on how to set up a build environment.

2.1.3 Project Build Macro Configuration

The macro configuration is part of the build configuration in `/source/$PROJECT/_config`. It can be added at the end after a `Macros:` line.

2.1.4 An OBS Package

An OBS Package is a sub-namespace below a project. It contains the specification of a single package build for all specified repositories.

2.2 The OBS Interconnect

The OBS interconnect is a mechanism to connect two OBS instances. All content, including sources and binary build results, will be available in the connecting instance. Unlike other methods the instances will also notify each other about changes.

2.3 Download on Demand Repositories (DoD)

2.3.1 Motivation

In a DoD repository external software repositories can be configured which are used for dependency resolution and where packages will be downloaded at build time. A DoD repository has some main advantages in comparison to binary import projects:

- less disk usage as only really required packages will be downloaded
- automatic package updates when new upstream releases are available
- simple to configure in project meta with no for shell access to repo servers

In download repotypes where package checksums can be verified (e.g. susetags, rpmmd and deb), we recommend that you use a mirror server URL in <download> in order to reduce traffic on the master server and configure a <master> with an **`https`** url and a **`sslfinger`** in order to avoid man in the middle attacks by peer verification.

2.3.2 XML Document Hierarchy

```
<project>
  <repository>
    <download>
      <master/> (optional)
      <pubkey/> (optional)
    </download>
  </repository>
</project>
```

2.3.3 The Daemon

The bs_dodup daemon periodically checks for new metadata in remote repositories. This daemon can be enabled for startup with the command

```
systemctl enable obsdodup.service
```

and can be started with

```
systemctl start obsdodup.service
```

2.3.4 The `download` Element

mandatory attributes:

- arch
- url
- repotype

2.3.5 The `master` Subelement

The **`<master>`** tag as shown in the rpmmd example below is optional but strongly recommended for security reasons.

Verification is supported in the following repotypes

- susetags
- rpmmd
- deb

This option could be defined by any valid URL (HTTP and HTTPS) to the origin of the repository but it is strongly recommended to use **`https`** with a **`sslfingerprint`** to bs_dodup possibility to verify its peer in order to avoid man-in-the-middle attacks. The download URL can be a mirror as we validate package checksums found in repo data.

You can easily query the SSL fingerprint of a remote server with the following command:

```
openssl s_client -connect <host>:<port> < /dev/null 2>/dev/null | openssl x509 -
fingerprint -noout
```

2.3.6 The pubkey Subelement

The pubkey element contains one or more GPG public keys in order to verify repository information but not packages. For an example, look at the repotype "deb" documentation below.

2.3.7 Repository Types

2.3.7.1 YAST Sources (susetags)

`Example:`

```
<project name="My::SuSE::CD">

  [...]

  <repository name="standard">
    <download arch="x86_64" url="http://mirror.example.org/path/to/iso"
 repotype="susetags" />
```

```
    <download arch="i586" url="http://mirror.example.org/path/to/iso"
 repotype="susetags" />
    <arch>x86_64</arch>
    <arch>i586</arch>
  </repository>
</project>
```

2.3.7.2 RPM Sources (rpmmd)

Example:

```
<project name="Fedora:Rawhide">

  [...]

  <repository name="standard">
    <download arch="x86_64" url="http://mirror.example.org/fedora/rawhide/x86_64/os"
 repotype="rpmmd">
      <master url="https://master.example.org/whereever/fedora/rawhide/x86_64/os"
 sslfingerprint="sha256:0a64..0303"/>
    </download>
    <download arch="i586" url="http://mirror.example.org/fedora/rawhide/i386/os"
 repotype="rpmmd">
      <master url="https://master.example.org/whereever/fedora/rawhide/i386/os"
 sslfingerprint="sha256:0a64..0303"/>
    </download>
    <arch>x86_64</arch>
    <arch>i586</arch>
  </repository>
</project>
```

2.3.7.3 Apt Repository (deb)

Apt supports two repository types, flat repositories and distribution repositories.

The download url syntax for them is:

- <baseurl>/<distribution>/<components>
- <flat_url>/.[/<components>]

You can specify multiple components separated by a comma.

An empty components string is parsed as "main".

Example:

```
<project name="Debian:8">

  [...]

  <repository name="ga">
    <download arch="x86_64" url="http://ftp.de.debian.org/debian/jessie/main"
 repotype="deb">
      <pubkey>
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1.4.12 (GNU/Linux)

[...]

      </pubkey>
    </download>
    <download arch="i586" url="http://ftp.de.debian.org/debian/jessie/main"
 repotype="deb">
      <pubkey>
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1.4.12 (GNU/Linux)

[...]

      </pubkey>
    </download>
    <arch>x86_64</arch>
    <arch>i586</arch>
  </repository>
</project>
```

2.3.7.4 Arch Repository (arch)

Be aware that there is currently no way to verify the origin of repository for Arch.

Example:

```
<project name="Arch:Core">

  [...]

  <repository name="standard">
    <download arch="x86_64" url="http://ftp5.gwdg.de/pub/linux/archlinux/core/os/x86_64"
 repotype="arch"/>
```

```
    <download arch="i586" url="http://ftp5.gwdg.de/pub/linux/archlinux/core/os/i686"
 repotype="arch"/>
    <arch>x86_64</arch>
    <arch>i586</arch>
  </repository>
</project>
```

2.3.7.5 Mandriva Repository (mdk)

Example:

```
<project name="Mageia:5">

  [...]

  <repository name="standard">
    <download arch="x86_64" url="http://mirror.example.org/Mageia/distrib/5/x86_64/media/
core/release" repotype="mdk"/>
    <download arch="i586" url="http://mirror.example.org/mirrors/Mageia/distrib/5/i586/
media/core/release" repotype="mdk"/>
    <arch>x86_64</arch>
    <arch>i586</arch>
  </repository>
</project>
```

2.4 Integrating External Source Repositories

2.4.1 Motivation

This chapter makes some recommendations how upstream resources can be integrated into the build process. SCM stands for source control management. git, subversion or CVS are concrete implementations of an SCM. The OBS itself comes also with an own SCM, but this is only intended to manage the files needed for packaging. However, you can add references to external SCM systems. The source service system will mirror the sources and provide it to the build systems.

OBS makes sure that you can access the sources of all builds also in the future, even when the upstream server delivers different or no content at all anymore. Using external SCM references has the following advantages:

- It is documented where a source comes from and how to create the archive.
- Working on the upstream sources can be done directly in local checkouts and changes can be tested via local builds before pushing to the SCM server.
- The sources can be stored incrementally and need less storage on the server.

2.4.2 Creating an Reference to an External SCM

External references are defined in _service files. The file can look like this:

```
<services>
  <service name="obs_scm">
    <param name="url">git://...</param>
    <param name="scm">git</param>
  </service>
  <service name="tar" mode="buildtime"/>
  <service name="recompress" mode="buildtime">
    <param name="file">*.tar</param>
    <param name="compression">xz</param>
  </service>
  <service name="set_version" mode="buildtime" />
</services>
```

The services do the following:

- `obs_scm`: mirrors the source. It stores it as a cpio archive, but for the build process this looks like a directory. It also stores additional information from the metadata to a file with obsinfo suffix.
- `tar`: creates a tar file from the directory
- `recompress`: applies a compression on the tar file
- `set_version`: reads the version from the obsinfo file and adapts the build descriptions to it.

Note that only the first service (`obs_scm`) runs on the OBS server. The other services run during the build process. They can also be replaced by any user by providing alternative implementations of them, or by writing their own service from scratch.

2.4.3 Working with Local Checkouts

Using **`osc build`** in any package with such a definition will do the same process locally. The only difference is that you get a local subdirectory with the SCM content. You can go inside and work as you are used to. Any changes inside will be used for your next local build, whether they were pushed to the upstream server or not. However, you need to push it upstream when you let the OBS server re-fetch the changes from upstream. The only way out would be to set the `obs_scm` service to mode disabled and upload your local archive.

2.4.4 Managing Build Recipes in a SCM

The `obs_scm` service allows you to export files next to the archive. You can specify one or more files using the `extract` parameter. Use it for your build recipe files.

3 Build Process

Each package build is created in a fresh environment. This is done to ensure that all dependencies are available and that every later build produces identical results.

3.1 Phases of a Build Process

All sources and binaries which are hosted inside Open Build Service are organized in projects. Projects host sources inside of OBS packages. The sources are build according to the repository configuration inside of the project.

3.1.1 Preinstall Phase

This phase depends on the type of the buildroot (building environment). OBS supports three different types of buildroots:

- chroot
- Xen
- KVM

In the preinstall phase, the OBS Worker creates a small base system (chroot or VM Image) with manually extracted packages (file system, coreutils, binutils, rpm/debutils, etc.), copies all necessary build requirements into the base system.

3.1.2 Install Phase

Depending on the chosen build root, the worker starts a Xen or KVM virtual machine or enters the build root. If this was successful, the install phase reinstalls all base packages from above and additionally all packages you have defined in your build recipe. After this phase the environment is ready to process the build recipe.

3.1.3 Package Build

Depending on the type of package, the buildroot executes different build commands:

- RPM-based distributions: **`rpmbuild`**
- Debian-based distributions: **`dpkg-buildpackage`**
- Arch Linux: **`pacman`**.

How the build continues depends on the quality and the type of your build recipe. In most cases, the source code will be compiled now and then be packed into the chosen package format.

To improve package quality, on RPM-based distributions there are additional security checks and a linter called **`rpmlint`**.

3.1.4 After the Build

The generated packages are taken from the worker, are signed by the OBS signer and are published to the Repository.

3.2 Identify a build

OBS is usally tagging each build with an identifier. This can be used to find the building OBS instance, the project, repository and exact source for a binary. This information is stored in some variable called DISTURL and is specified as **`obs://$OBS_INSTANCE/$PROJECT/$REPOSITORY/$SOURCE_REVISION-$PACKAGE(:$FLAVOR)`** . Note that the last $:FLAVOR part is optional and exists only when the package was build using the multibuild feature. The source specified via the DISTURL can be accessed by pasting the URL into the search interface of the OBS web interface. Or use the command line tool to check it out:

```
# osc checkout $DISTURL
```

You need to go to the right OBS instance as this is not handled automatically yet.

3.2.1 Read DISTURL from a rpm

Rpm binaries contain the DISTURL as tag. It can be read from the rpm database for installed rpms and also from the rpm binaries itself.

```
# rpm -q --qf '%{DISTRL}\n' $rpm
```

3.2.2 Read DISTURL from a container

Containers store the DISTURL as label. You will see only the DISTURL from the highest layer via

```
# docker inspect --format '{{.Config.Labels}}' $image_id
```

The disturl is always set via the key 'org.openbuildservice.disturl'.

4 Source Management

4.1 Find Package Sources

OBS is adding information to each created package about the origin of the sources. This information is stored in the **DISTURL** tag of an rpm, which can be displayed as follows:

```
rpm -q --queryformat '%{DISTURL}\n' glibc
rpm -q --queryformat '%{DISTURL}\n' -p glibc-2.1.0-1.i586.rpm
```

The disturl can look like this: **obs://build.opensuse.org/openSUSE:Factory/standard/80d21fdd2299302358246d757b4d8c4f-glibc** It always starts with `obs://`. The second part is the name of the build instance, which usually also hosts the Web UI. Next comes the project name and the repository name where the binary got built. Last part is the source md5 sum and the package name.

The disturl can also be entered in the search field of the web interface of the build service.

5 Request And Review System

The OBS comes with a generic request system where one party can ask another to complete a certain action. This can be, for example, taking source changes, granting maintainer rights or deleting a package. Requests are also used deal with more complex workflows.

A request is an object in the database. It can be accessed via the /request API route. osc and the web interface can show and process these requests. There are also interfaces to show the requests which should be handled for a certain user.

5.1 What a request looks like

A request is an object in the database. It can be accessed via the /request API route. Main parts of the request are

- state: The state tells if the request still needs to processed or has been handled already and how.
- actions: these are the changes which will be applied when accepting the request.
- reviewer: reviewer can be added automatically at request creation time or manually by any involved party. Usually all of them should approve the request before it will be accepted. However, the target can ignore that and accept anyway optionally.
- description: an explanation of why the actions should be done.
- history: a history about state changes of the request.
- accept_at: the request will get accepted automatically after the given time. Such a request can only be created when having write permissions in the target. Automatic cleanup requests created by Admin user are using this.

Requests can only be accepted or rejected in their entirety. Therefore, it can make sense to have multiple actions in one request if changes should be applied in one transaction. For example, submitting a new package and removing an old instance: Do either both or nothing. This implies that the person accepting the request must have write access in all targets or they will not be allowed to accept the request.

5.1.1 Action Types

Actions always specify some target. This can be either a project or a package. Further information depend on the action type. The following gives an overview, for details, see the XML schema for requests.

5.1.1.1 `submit`

A submit action will transfer sources from one package to another package. Usually a submit request will refer to a specific revision in the source, but it does not have to. If no revision is specified, then the current revision at the time of acceptance will be used. This should be avoided when relying on complex reviews during the request process. Hence, it is recommended to identify a specific version in your submitrequest (`osc submitrequest -r 42 ...`).

The submit action can support options to update the source or even to remove the source. Tools like osc are applying the cleanup rule by default when submitting from a default user home branch project.

5.1.1.2 `delete`

A delete action can request removal of a project or package instance.

5.1.1.3 `add_role`

An add_role requests a specific role for a given user or group to the target. For example, one could use this to ask for maintainer rights, or to become a default reviewer.

5.1.1.4 `set_bugowner`

set_bugowner is similar to add_role, but removes all other bugowner roles in the target. This happens to have a unique identifier to be used when assigning bug reports in external tools like Bugzilla.

5.1.1.5 `change_devel`

can be used to update the devel package information in the target.

5.1.1.6 `maintenance_incident`

Official request to open a maintenance incident for official support products. This create by developers who want to start an official maintenance process. Find detail in the maintenance chapter about this. If accepted, a new maintenance incident project is created and package sources listed are copied there. All sources of all actions in one request will be merged into the same maintenance incident.

5.1.1.7 `maintenance_release`

Is used to release a ready maintenance update. Unlike maintenance_incident or submit, sources and binaries are copied without a rebuild. Details can be found in maintenance chapter.

5.1.1.8 `group`

Deprecated. Was never in a released OBS version. It is not allowed to be used anymore.

5.1.2 Request states

- new: The default value for newly created requests. Everybody involved in the specified targets can see the request and accept or decline it.
- accepted: The request has been accepted and the changes applied. history files have a reference to this request.
- declined: The request has been reviewed and not (yet) been accepted by the target. This is often used to ask for some more information from the submitter, since declined requests remain active, returning to the submitter's active request queue (that is, the submitter will need to take action now).
- revoked: The submitter has taken back their request. The request is considered to be inactive now.
- superseded: This request is obsolete due to a new request. The request is considered to be inactive now. The superseding request is linked in this request.
- review: There are still open reviews inside of the request. Nobody has declined it yet. The request is not yet visible to the target by default. The state will change automatically to new when all reviewers accept.

5.1.3 Reviewers

Reviews can be done by users, groups, projects or packages. Review by project or package means that any maintainer of them is asked for reviews. This is handy to avoid the need to figure who actually is a maintainer of a certain package. Also, new maintainers of a package will see requests in case the old maintainer did not handle them.

5.1.3.1 Manual added reviews

Reviewers can be added manually by anyone involved in a request. This can be used to hand over a review. In that situation the new reviewer needs to be added and the own review needs to be accept. The request becomes declined when any of the reviewers are declining the request.

5.1.3.2 Automatic added reviews

Project and package objects can have users or groups with a reviewer role. They are added automatically to a request as reviewer when a request is created which has them as target. In case the project and package bother specify reviewer all of them are added to the request.

5.1.4 Request creation

The API is doing a number of checks on request creation time. In case a target is not specified it tries to set it according to the linked package. If an entire project is specified as source it expands it to refer all packages inside. This means it is replacing one action with multiple. When using the addrevision parameter it does also add the current revision of the package source to the action. This makes it easy to create new requests with little logic in the client.

5.1.5 Request operations

Requests can be modified only in very limited ways after creation. This is to avoid that reviewers reviewed a request but the nature of the requests is changing afterwards. Valid operations on a request are:

- diff: does not modify the request, just shows source modifications wanted by the request
- changestate: to change the state of the request, for example to accept it.

- changereviewstate: to change the state of a review inside of a request.
- addreviewer: add further reviewer to a request

6 Image Templates

Image templates are pre-configured image configurations. The image templates page (https://build.opensuse.org/image_templates) provides a list of these templates. Users can clone these templates and further configure them as they like.

6.1 Structure of Image Templates

As mentioned image templates are essentially pre-configured KIWI (http://opensuse.github.io/kiwi/) image configurations. As any KIWI configuration they usually contain a tarball containing image sources, a config.sh file and the KIWI configuration XML file.

In addition, you can define an icon for your image templates by adding graphical image (for example, PNG, JPG) to your template sources and name it **`_icon`**. If that file exists, it will be used as icon for your image on the image templates page.

Note

For more information about KIWI images, see *Book "User Guide", Chapter 1 "Supported Build Recipes and Package Formats", Section 1.5 "KIWI Appliance"*.

6.2 Adding Image Templates to/Removing Image Templates from the Official Image Template Page

The image templates page lists templates per project. New templates get added by setting the **`OBS:ImageTemplates`** attribute to a project. Any package container belonging to a project with that attribute will be shown on the template page.

Only admins can add / remove the OBS:ImageTemplates attribute from a project.

6.3 Receiving Image Templates via Interconnect

If your OBS instance is connected to a remote instance via interconnect, OBS will fetch image templates from the remote instance and present it on the image templates page. They appear below the local templates.

For more information about interconnects, see *Book "Administrator Guide", Chapter 3 "Administration", Section 3.2 "Managing Build Targets"*.

7 Multiple Build Description File Handling

7.1 Overview

A package source may contain multiple build description files. They can be used depending on the base distribution, the repository name or for different configurations. These mechanics can be also combined.

The right build description file gets picked by filtering. The build will not start when either no file matches or multiple candidates exist. The filtering happens with the following steps:

1. Based on the package build format of the based distributions. RPM-based distributions will use spec files for example.
2. Based on the file name of the file before the suffix. It is not important as long as just one file exists, but it has to match when multiple files exist. The name is defined by the build container name, which is either defined in a _multibuild directive file or is the source package name.
3. Specific files can be created to be built for a specific repository. Append the repository name of the build container behind the package name with a -. For example **`hello-openSUSE_13.2.spec`**.

7.2 How Multibuild is Defined

Use the _multibuild directive to build the same source in the same repository with different flavors. This handy to define all flavors in one place without the need to maintain packages with local links. This allows also to transfer all sources including a possible changed flavor from one project to another with a standard copy or submit request.

The _multibuild file lists all build container names, each of them will be build as usual for each defined repository and each scheduler architecture.

For example, inside the **`kernel`** source package we can build both **`kernel-source`** and **`kernel-obs-build`** packages by listing them inside the file.

Multibuild packages are defined with the **`_multibuild`** directive file in the package sources.

The **`_multibuild`** file is an xml file. For example:

```
<multibuild>
```

```
  <flavor>kernel-source</flavor>
  <flavor>kernel-obs-build</flavor>
</multibuild>
```

Build description files are needed for each of them for each package (eg. kernel-source.spec or kernel-obs-build.dsc) inside of the sources. There will be another build in case there is also a matching file for the source package container name, otherwise it will turn into an “excluded” state.

8 Maintenance Support

This chapter explains the setup and workflow of a maintenance update in the openSUSE way. However, this should not be limited to openSUSE distribution projects but be usable anywhere (the entire workflow or just parts of it).

The goal of the OBS maintenance process is to publish updates for a frozen project, in this example an entire distribution. These updates need to be approved by a maintenance team and the published result must contain documentation about the changes and be applicable in the easiest way by the users. The result is a package repository with additional information about the solved problems and defined groups of packages to achieve that. Binary delta data can also be generated to reduce the needed download size for the clients.

8.1 Simple Project Setup

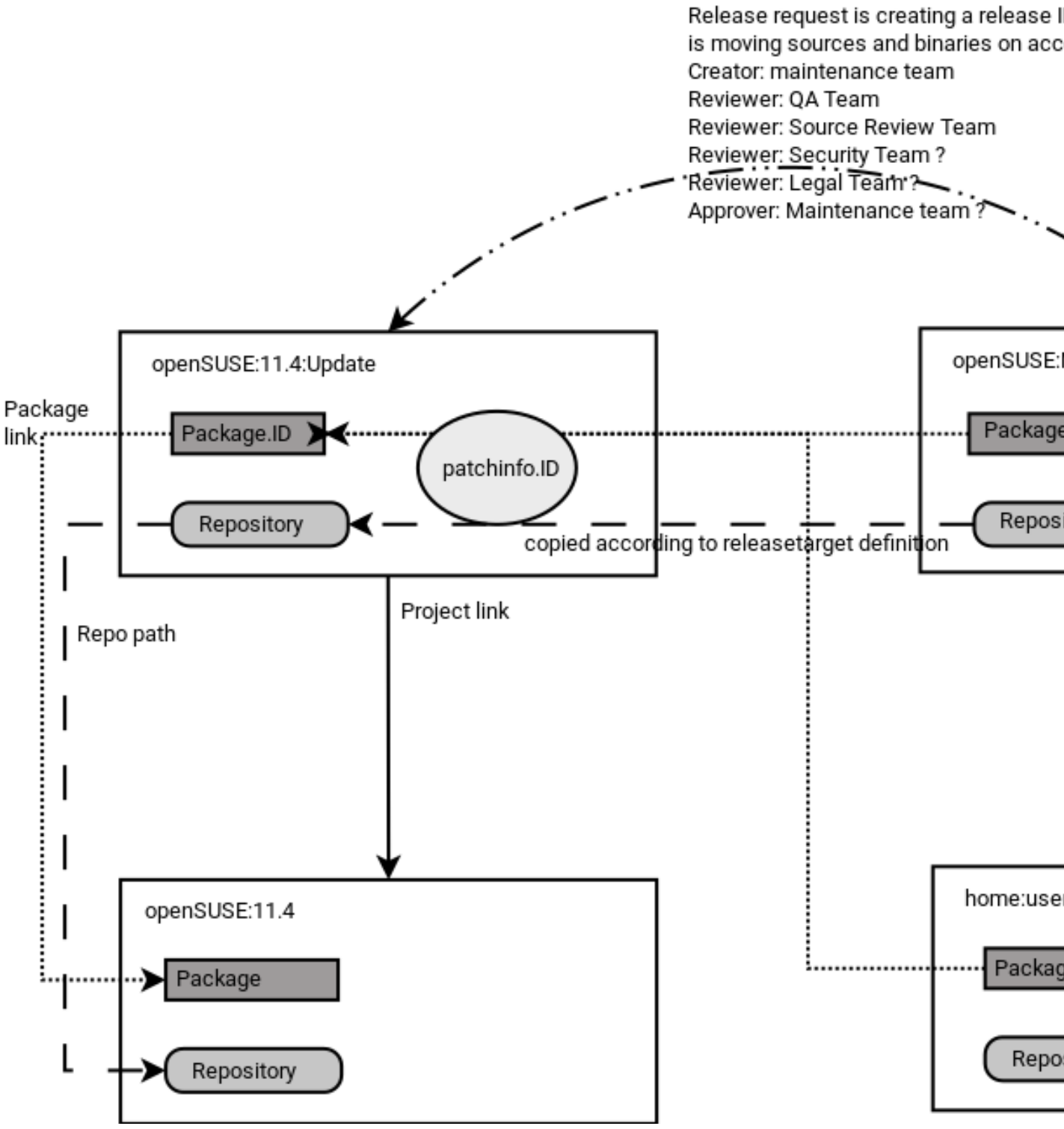


FIGURE 8.1: SIMPLE PROJECT SETUP

This figure gives an overview about the project setup and general workflow for a single package and single maintained distribution. It shows the "openSUSE:11.4" project, which is considered to be frozen and not changing at all anymore. The "openSUSE:11.4:Update" projects hosts all officially released updates. It does not build any binary, just gets it sources and binaries from the maintenance incident project via the release process. The incident project is named "openSUSE:Maintenance:IDxxx" in this example, which is under control of the maintenance team. Official updates get built and reviewed here. QA teams are also testing the binaries from here. However, a user can prepare it in the same way in their project and start the maintenance process via doing a "maintenance" request.

- openSUSE:11.4 is the *GA Project* in this example. Frozen and not changing anymore.
- openSUSE:11.4:Update is the *Update Project* to release official updates.
- openSUSE:Maintenance is the *Maintenance Project*.
- openSUSE:Maintenance:IDxxx is the *Incident* project.

8.2 Using the Maintenance Process

This describes all required steps by all involved persons from preparing to releasing a maintenance update.

8.2.1 Workflow A: A Maintainer Builds an Entire Update Incident for Submission

A user is usually starting to prepare an update by creating a maintenance branch. This is typically done by creating an own maintenance project. Usually multiple released products are affected, so the server can find out which one are maintained by a given source package name, in this example for glibc including checkout via

```
osc mbranch glibc
osc mbranch --checkout glibc
```

This is equivalent to the API call `/source?cmd=branch&package=glibc`.

It is also possible to branch only one defined version, if it is known that only one version is affected. In this example the openSUSE:12.1 version:

```
osc branch --maintenance openSUSE:12.1 glibc
```

```
osc branch -M -c openSUSE:12.1 glibc
```

NOTE: both branch commands do support the --noaccess parameter, which will create a hidden project. This may be used when a not yet publicly known security issue is get fixed.

Afterwards the user needs to do the needed modifications. Packages will be built and can be tested. Afterwards they may add information about the purpose of this maintenance update via

```
osc patchinfo
```

If the source changes contain references to issue trackers (like Bugzilla, CVE or FATE) these will be added to the _patchinfo file.

The server will create a full maintenance channel now, in case the user wants to test this as well. After the user has tested, they have to create a maintenancerequest to ask the maintenance team to accept this as an official update incident:

```
osc maintenancerequest
```

On accepting this request all sources of the entire project will get copied to the incident project and be rebuild. The origin project gets usually removed (based on the request cleanup options).

8.2.2 Workflow B: Submitting a Package Without Branching

You may submit a package source from a project which is not prepared as maintenance project. That works via the maintenancerequest mechanism by specifying one or more packages from one project. As a consequence it means also that the first testable build will happen in the maintenance incident project. Also, the maintenance team need to write the update information on their own.

```
osc maintenancerequest [ SOURCEPROJECT [ SOURCEPACKAGES RELEASEPROJECT ] ]
```

The following example is submitting two packages (kdelibs4 and kdebase4) from the project KDE:Devel project as update for openSUSE:12.1

```
osc maintenancerequest KDE:Devel kdelibs4 kdebase4 openSUSE:12.1
```

Note: Specifying an Existing Incident

It is also possible to specify an existing incident as target with the `--incident` parameter. The packages will then be merged into the existing incident project.

8.2.3 Workflow C: Process Gets Initiated By the Maintenance Team

The maintenance team may start the process (for example because a security issue was reported and the maintenance team decided that a fix is required). In this case the incident gets created via the Web UI or via the API call:

`osc createincident [PROJECT]`

`osc api /source/PROJECT?cmd=createmaintenanceincident`

`osc api /source?cmd=createmaintenanceincident&attribute=OBS:Maintenance`.

To document the expected work the creation of a patchinfo package is needed. This can be done via

`osc patchinfo [PROJECT]`

It is important to add Bugzilla entries inside of the _patchinfo file. As long these are open Bugzilla entries, the bug assignee will see this patchinfo on their "my work" Web UI and osc views, so they knows that work is expected from them.

8.2.4 Maintenance Incident Processing

The maintenance incidents are usually managed by a maintenance team. In case the incident got started by a maintainer a maintenance request is targeted towards the defined maintenance project, in our example this is openSUSE:Maintenance. The defined maintainer and reviewers in this project need to decide about this request. In case it gets accepted, the server is creating a subproject with a unique incident ID and copies the sources and build settings to it. The origin project will get removed usually via the cleanup option. This maintenance project is used to build the final packages.

If the maintenance team decides to merge a new maintenance request with an existing incident, they can run the `osc rq setincident $REQUESTID $INCIDENT` before accepting the request.

The maintenance team may still modify them or the patchinfo data at this point. An outside maintainer can still submit changes via standard submit request mechanism, but direct write permissions are not granted. When the maintenance people are satisfied with the update, they can create a request to release the sources and binaries to the final openSUSE:11.4:Update project.

```
osc releaserequest
```

The release request needs to specify the source and target for each package. In case just the source package or project is specified the api is completing the request on creation time. It is using this based on the source link target of each package and the release information in the repository definitions.

8.2.5 Incident Gets Released

The release process gets usually started via creating a release request. This sets all affected packages to the locked state, which means that all commands for editing the source or triggering rebuilds are not allowed anymore.

The release request typically needs to be approved by QA and other teams as defined in the Update project. In case something gets declined, the necessary changes need to be submitted to the maintenance project and a new release request has to be created.

A unique release ID will be generated and become part of the updateinfo.xml file in the target project on release event. This ID is different from the incident ID and is usually in the style of "YEAR-COUNTER". The counter is strictly increasing on each release. In case of a re-release of the same incident a release counter will be added.

A different naming scheme can be defined via the OBS:MaintenanceIdTemplate attribute value. The release will move all packages to the update project and extend the target package name with the incident ID. Binaries will be moved as well without modification. The exception is the updateinfo.xml which will be modified by replacing its incident id with the release id.

8.2.6 Incident Gets Reopened and Re-Released

An update should not, but may have an undetected regression. In this case the update needs a re-release. (If another problem shall be fixed a new incident should be created instead.)

If the current update harms the systems, the maintenance team may decide to take it back immediately. It can be done by removing the patchinfo.ID package container in the Update projects. This will create a new update channel without this update.

To re-open a release incident project, it must get unlocked and marked as open again. Unlocking can be done either via revoking a release request or via explicit unlocking the incident. The explicit unlock via osc: **`osc unlock INCIDENT_PROJECT`** is also triggering a rebuild to ensure to

have higher release numbers and adding the "trigger=maintenance" flags to the release target definitions. Afterwards the project can be edited again and also gets listed as running incident again.

8.2.7 Using Custom Update IDs

The used string of update IDs can be defined via the OBS:MaintenanceIdTemplate attribute value of the master maintenance project.

8.3 OBS Internal Mechanisms

OBS is tracking maintenance work and can be used as a database for future and past updates.

8.3.1 Maintenance Incident Workflow

A maintenance incident is started by creating the incident project, either via a developer request or by the maintenance team.

1. Incident project container is created. This is always a sub project to the maintenance project. A unique ID (counter) is used as subproject name. Build is disabled by default project wide.
2. Default content for an incident is added via branch by attribute call:
   - Package sources get added based on given package and attribute name from all existing project instances. The package name is extended by the source project name to allow multiple instances of same package in one project. Source revision links are using the xsrcmd5 to avoid that other releases will affect this package instance.
   - Build repositories are added if missing. All repositories from all projects where the package sources gets branched from are used. The build flags in the package instances gets switched on for these.
   - A release target definition is added to the repository configuration via additional releasetarget element. The special release condition "maintenance" gets defined for this.
3. Fixes for the packages need to get submitted now.

4. A patchinfo file need to get added describing the issue.
5. OBS server is building packages according to the sources and update information according to the patchinfo data.
6. one or more release requests get created. It does also set the project to "freeze" state by default, this means no source changes are possible anymore and all running builds get canceled.
7. Usually the request is in review state with defined reviewers from the release project. All reviewers need to review the state in the incident project.
8. Request changes into state "new" when all reviewers accepted the release request.
9. The release happens on accepting the request by the maintainers of the release project.
    - All package sources and binaries get copied into a package container where the package name gets extended by the incident number.
    - A main package gets created or updated, it just contains a link to the current incident package. Eg glibc points to glibc.42. The purpose of this main package is to have a place to refer to the current sources of a package.
    - The release target condition = maintenance gets removed.
    - The updateinfo.xml gets updated with the existing or now created unique updateinfo ID.
    - The server will update the repository based on all existing binaries.
10. OPTIONAL: A maintenance coordinator may remove the release by removing the package instances inside the release project. The source link has to be fixed manually. (We may offer a function for this).
11. OPTIONAL: A maintenance incident can be restarted by
    - Removing the lock flag.
    - Adding again the condition = maintenance attribute to the release target which requires a re-release.

NOTE: The step 1 and 2 may be done via accepting an incident request instead.

8.3.2 Searching for Incidents

The Web UI shows the running and past incidents when going to the maintenance project (openSUSE:Maintenance in our example). It shows the open requests either for creating or release an incident. Also, the open incidents, which are not yet released are visible.

All users need usually just to visit their "my work" screen in Web UI or osc to see requests or patchinfos where actions of them are expected: **`osc my [work]`**

The following items list some common ways to search for maintenance incidents via the api:

- A developer can see the work to be done by them via searching for patchinfos with open Bugzilla entries:

```
/search/package?match=([kind='patchinfo' and issue/[@state='OPEN' and owner/
@login='$USER_LOGIN']])
```

- A maintenance coordinator can see requests for doing a maintenance release via searching for open requests with maintenance_incident action against the maintenance project. They are visible in the Web UI request page of that project or via

```
/search/request?match=(state/@name='new') and action/@type='maintenance_incident'
 and action/target/@project='openSUSE:Maintenance')
```

- A maintenance coordinator can see open incidents via searching for incidents project repositories which have a release target with maintenance trigger. Note: this search result is showing all repositories of a matching project.

```
/search/project?match=(repository/releasetarget/@trigger='maintenance')
```

- A maintenance coordinator can see updates which currently are reviewed (for example by a QA team) via

```
/search/request?match=(state/@name='review') and action/@type='maintenance_release')
```

- A maintenance coordinator can see updates ready to release via searching for open requests with maintenance_release action.

```
/search/request?match=(state/@name='new') and action/@type='maintenance_release')
```

8.4 Setting Up Projects for a Maintenance Cycle

8.4.1 Defining a Maintenance Space

An OBS server is using by default a maintenance space defined via the OBS:Maintenance attribute. This must get created on a project where maintenance incident projects should get created below. This project is also defining the default maintenance maintainers and reviewers in its ACL list.

It is possible to have multiple and independent maintenance name spaces, however the maintenance request must be created against this other namespace manually or using a different attribute.

8.4.2 Maintained Project Setups

Maintained projects must be frozen, this means no changes in sources or binaries. All updates will be hosted in the defined update project. This project gets defined via the OBS:UpdateProject attribute which must contain a value with the update project name. In addition to this, an attribute to define the active maintenance should also be defined, by default the OBS:Maintained attribute. The osc mbranch command will take packages from this project as a result.

The Update project should be defined as build disabled as well. Also define a project link to the main project and at least one repository building against the main project.

8.5 Optional Channel Setup

Channels are optional definitions to publish a sub-set of binaries into own repositories. They can be used to maintain a larger amount of packages in a central place, but defining to published binaries with an independent workflow which requires an approval for each binary.

8.5.1 Defining a Channel

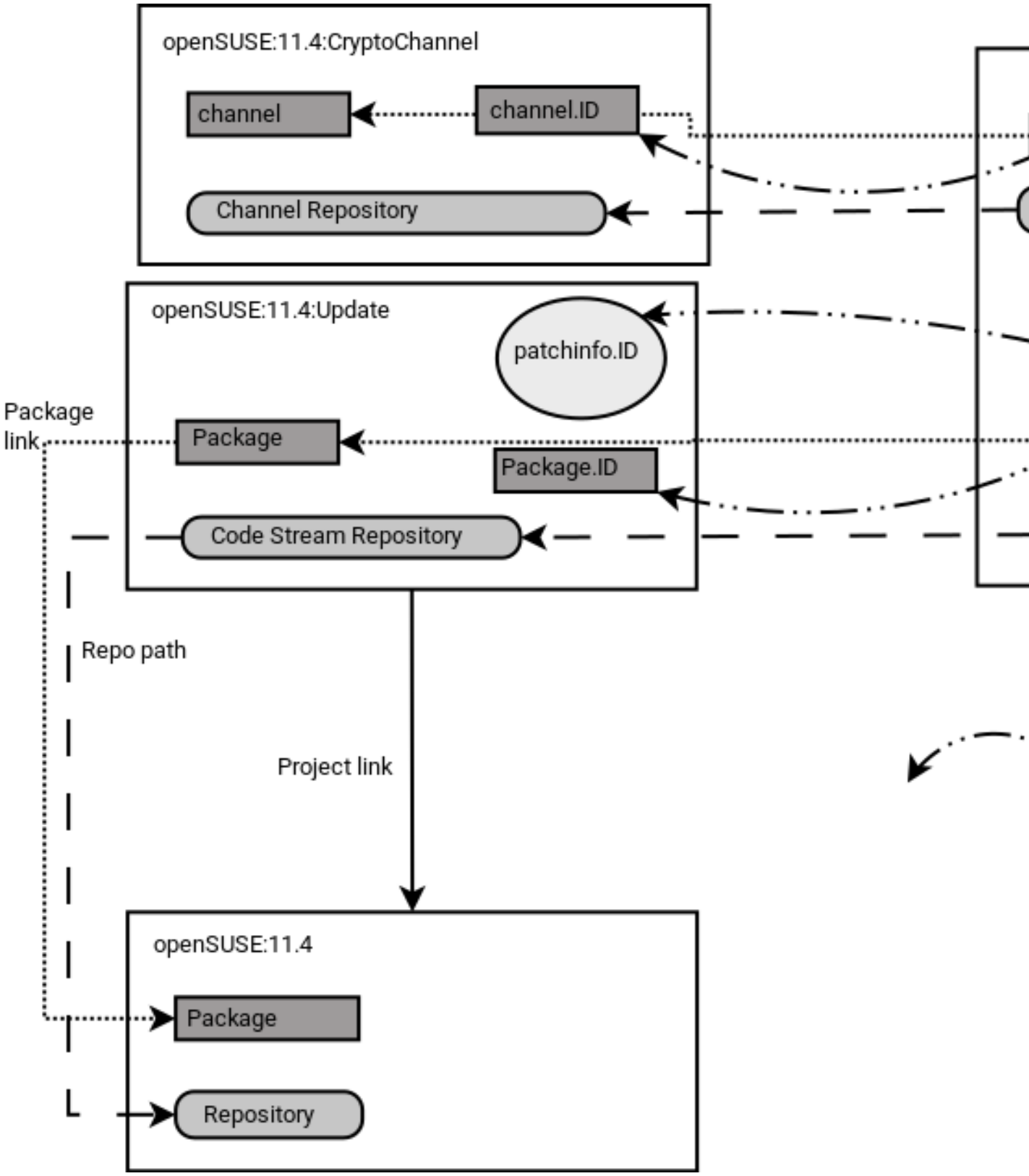
Channels get defined and maintained in an xml file inside of a package source. The file name of these lists must be _channel.

The file may contain a list of targets where binaries gets released to.

8.5.2 Using Channels in Maintenance Workflow

Channel definitions for existing packages do affect incident projects. Matching channel packages get automatically branched inside and additional repositories for the channels are created. The server will build the channel package by aggregating the binary packages into the channel repositories.

The _channel files can be modified inside of the incident project if needed. This can be necessary when binary packages get renamed or added with this update. The modification will be part of the maintenance release request as simple submit actions.


openSUSE:11.4:CryptoChannel
channel
channel.ID
Channel Repository
openSUSE:11.4:Update
patchinfo.ID
Package
link
Package
Package.ID
Code Stream Repository
Repo path
Project link
openSUSE:11.4
Package
Repository

This example shows the setup where selected binary packages get released also to a defined channel. The openSUSE:11.4:SecurityChannel project contains a _channel definition inside of the channel package. This one gets branched as well into the incident in case a matching channel does exist. Also, the additional repository gets added. The resulting binaries will be transfer via a release request to the code stream project (openSUSE:11.4:Update) and the special channel project.

9 Binary Package Tracking

Products and updates to them are often officially supported by a company. To allow giving such support, there is binary package tracking. This feature allows checking which exact version of a package was shipped at what time. This feature is often important for release managers, maintenance engineers, QA engineers and supporters.

OBS can track these binary packages and offer a database to search them.

9.1 Which Binaries Are Tracked?

All binaries which are released into projects providing `kind=maintenance_release` are tracked. In addition to that, the OBS administrator can configure additional projects via the `packtrack` setting in `BSConfig.pm`.

9.2 What Data Is Tracked?

In short the information to identify a binary, its building place and timestamps are tracked. In addition to that also information about possible successor versions or if the binary got removed in the meantime. If products do reference the repositories the search interface offers also a listing of products which are supposed to use it. Either as part of the product media itself or in one of its update repositories.

9.2.1 Binary Identifier

A binary is identified by the following information which is extracted from components of the file path of the binary:

- **Repository:** Where is the binary hosted?
- **Name:** Name of the binary file
- **Epoch:** The epoch version (optional, usually not used)
- **Version:** The version
- **Release:** The release number

- **Architecture:** The hardware architecture
- **Medium:** Name of the medium (exists only for product builds)

9.2.2 Binary Information

Additional information about a binary is information which gets updated when a binary gets added or replaced.

- operation, got the binary added, removed or modified
- publish time, aka the time when the repository gets published by OBS. This is not the same time as when the release action got invoked.
- build time
- obsolete time, exists only when a binary gets removed or replaced
- supportstatus, meta information about the level of support which is granted for the binary at the time of releasing it.
- updateinfo id from rpm-md repository
- maintainer of the binary who has prepared this update
- disturl, the exact identifier to the source and build repository

9.2.3 Product information

Additional information about products referencing to this binary.

- updatefor: the listed products do reference the repository as update channel.
- product: exists when the binary was part of a product medium

9.3 API Search Interface

The search is provided via the generic XPath search interface. It is provided below the paths:

- /search/released/binary/id : short form, just listing the matched binary identifiers
- /search/released/binary : long form, provides all other tracked information as described above

9.3.1 Examples

To find the latest version of given glibc-devel binary in all products. Skipping old and revoked versions:

```
/search/released/binary?match=@name"=glibc-devel"+and+obsolete[not(@time)]
```

To find a specific version by given updateinfo id. This ID is visible in the update tools to the end user:

```
/search/released/binary?match=updateinfo/@id="OBS-2014-42"
```

To find a specific version by given disturl. Used to find all affected products by a certain build of a binary:

```
/search/released/binary?match=disturl="obs://..."
```

When got the specific package version got released the first time:

```
/search/released/binary?match=@name='kernel-default'+and+@version='1.0'+and
+@release='1'+and+@arch='i586'+and+supportstatus='l3'+and+operation='added'
```

All binaries in a given repository:

```
/search/released/binary?match=repository/[@project='BaseDistro3'+and
+@name='BaseDistro3_repo']
```

All binaries part of a product release:

```
/search/released/binary?match=product/[@project='openSUSE'+and+@name='openSUSE'+and
+(@arch='x86_64'+or+not(@arch))]
```

All binaries part of the update repositories of a product:

```
/search/released/binary?match=updatefor/[@project='openSUSE'+and+@product='openSUSE'+and
+(@arch='x86_64'+or+not(@arch))]
```

All binaries part of the update repositories of a versioned product:

```
/search/released/binary?match=updatefor/[@project='openSUSE'+and+@product='openSUSE'+and
+@version='13.2']
```

All binaries part of the update repositories of a versioned product (enterprise style):

```
/search/released/binary?match=updatefor/[@project='openSUSE'+and+@product='openSUSE'+and
+@baseversion='12'+and+@patchlevel='1']
```

10 Administration

This chapter describes the components of an OBS installation and the typical administration tasks for an OBS administrator.

This chapter is not intended to describe special installation hints for a certain OBS version. Refer to the OBS download page (http://openbuildservice.org/download/) for that.

10.1 OBS Components

The OBS is not a monolithic server: it consists of multiple daemons that perform different tasks.

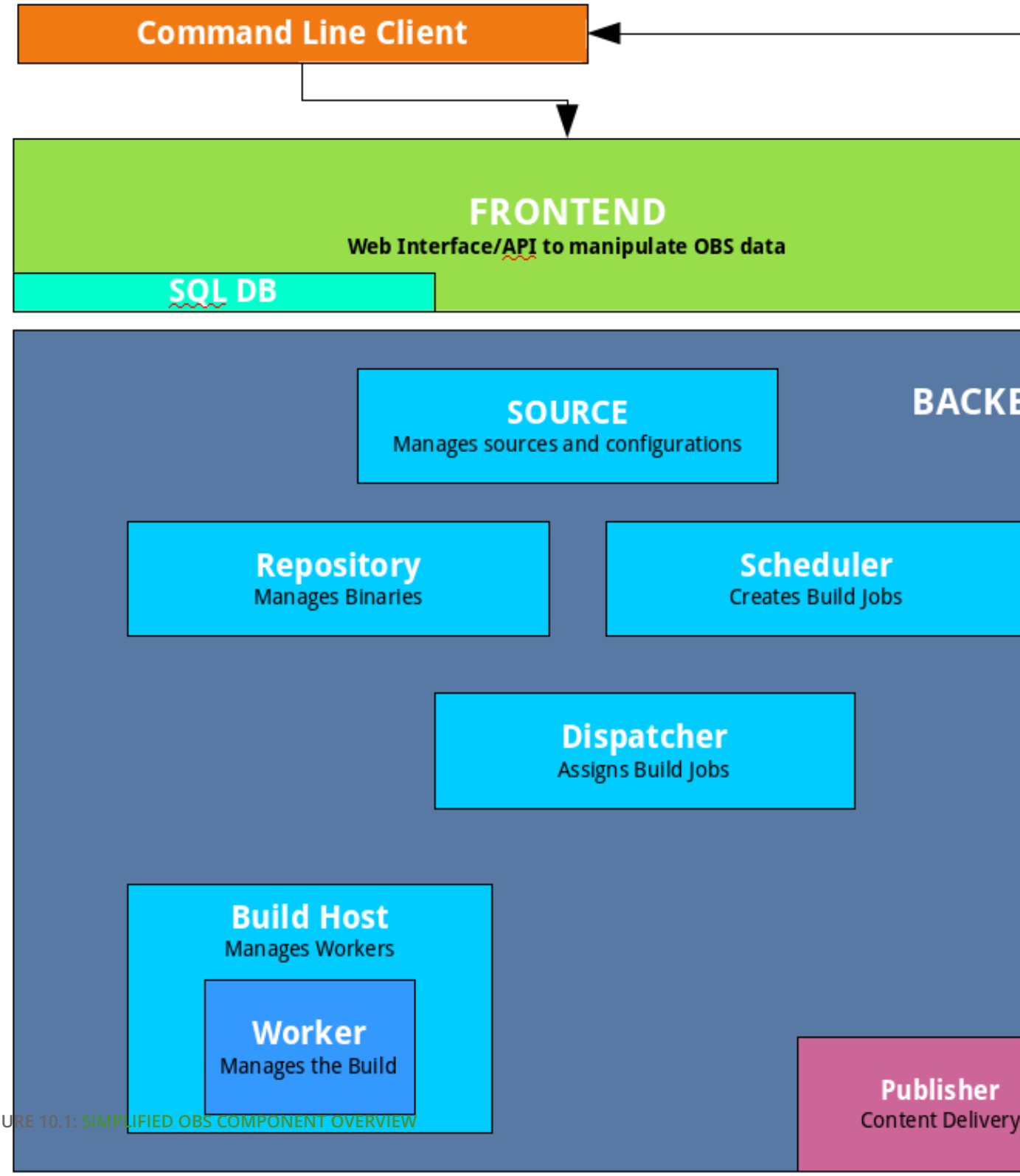


FIGURE 10.1: SIMPLIFIED OBS COMPONENT OVERVIEW

10.1.1 Front-end

The OBS Front-end is a Ruby on Rails application that manages the access and manipulation of OBS data. It provides a web user interface and an application programming interface to do so. Both can be used to create, read, update and delete users, projects, packages, requests and other objects. It also implements additional sub-systems like authentication, search, and email notifications.

10.1.2 Back-end

The OBS Back-end is a collection of Perl applications that manage the source files and build jobs of the OBS.

10.1.2.1 Source Server

Maintains the source repository and project/package configurations. It provides an HTTP interface, which is the only interface to the Front-end. It may forward requests to other back-end services. Each OBS installation has exactly one Source Server. It maintains the "sources", "trees" and "projects" directories.

10.1.2.2 Repository Server

A repository server provides access to the binaries via an HTTP interface. It is used by the front-end via the source server only. Workers use the server to register, request the binaries needed for build jobs, and store the results. Notifications for schedulers are also created by repository servers. Each OBS installation has at least one repository server. A larger installation using partitioning has one on each partition.

10.1.2.3 Scheduler

A scheduler calculates the need for build jobs. It detects changes in sources, project configurations or in binaries used in the build environment. It is responsible for starting jobs in the right order and integrating the built binary packages. Each OBS installation has one scheduler per available architecture and partition. It maintains the "build" directory.

10.1.2.4 Dispatcher

The dispatcher takes a job (created by the scheduler) and assigns it to a free worker. It also checks possible build constraints to verify that the worker qualifies for the job. It only notifies a worker about a job; the worker itself downloads the required resources. Each OBS installation has one dispatcher per partition (one of which is the master dispatcher).

10.1.2.5 Publisher

The publisher processes "publish" events from the scheduler for finished repositories. It merges the build result of all architectures into a defined directory structure, creates the required meta-data, and optionally syncs it to a download server. It maintains the "repos" directory on the back-end. Each OBS installation has one publisher per partition.

10.1.2.6 Worker

The workers register with the repository servers. They receive build jobs from the dispatcher. Afterwards they download sources from source server and the required binaries from the repository server(s). They build the package using the build script and send the results back to the repository server. A worker can run on the same host as other services, but most OBS installations have dedicated hardware for the workers.

10.1.2.7 Signer

The signer handles signing events and calls an external tool to execute the signing. Each OBS installation usually has one signer per partition and one on the source server installation.

10.1.2.8 Warden

The warden monitors the workers and detects crashed or hanging workers. Their build jobs will be canceled and restarted on another host. Each OBS installation can have one Warden service running on each partition.

10.1.2.9 Download on Demand Updater (dodup)

The download on demand updater monitors all external repositories which are defined as "download on demand" resources. It polls for changes in the metadata and re-downloads the metadata as needed. The scheduler will be notified to recalculate the build jobs depending on these repositories afterwards. Each OBS installation can have one dodup service running on each partition.

10.1.2.10 Delta Store

The delta store daemon maintains the deltas in the source storage. Multiple obscpio archives can be stored in one deltastore to avoid duplication on disk. This service calculates the delta and maintains the delta store. Each OBS installation can have one delta store process running next to the source server.

10.1.3 Command Line Client

The Open Build Service Commander (osc) is a Python application with a Subversion-style command-line interface. It can be used to manipulate or query data from the OBS through its application programming interface.

10.1.4 Content Delivery Server

The OBS is agnostic about how you serve build results to your users. It will just write repositories to disk. But many people sync these repositories to some content delivery system like MirrorBrain (http://mirrorbrain.org/).

10.1.5 Requirements

We highly recommend, and in fact only test, installations on the SUSE Linux Enterprise Server (https://www.suse.com/products/server/) and openSUSE (http://www.opensuse.org) operating systems. However, there also are installations on Debian and Fedora systems.

The OBS also needs a SQL database (MySQL or MariaDB) for persistent and a memcache daemon for volatile data.

10.2 OBS Appliances

This chapter gives an overview over the different OBS appliances and how to deploy them for production use.

10.2.1 Server Appliance

The OBS server appliance contains a recent openSUSE distribution with a pre-installed and pre-configured OBS front-end, back-end and worker. The operating system on this appliance adapts to the hardware on first boot and defaults to automatic IP and DNS configuration via DHCP.

10.2.2 Worker Appliance

The OBS worker appliance includes a recent openSUSE distribution and the OBS worker component. The operating system on this appliance adapts to the hardware on first boot, defaults to automatic IP and DNS configuration via DHCP and OBS server discovery via SLP.

10.2.3 Image Types

There are different types of OBS appliance images.

TABLE 10.1: APPLIANCE TYPES

File Name Suffix	Appliance for
.vdi	VirtualBox (https://www.virtualbox.org/).
.vmdk	VMware (http://www.vmware.com/) Workstation and Player. **Note** Our VirtualBox images are usually better tested.
.qcow2	QEMU/KVM (http://qemu.org).
.raw	Direct writing to a block device

File Name Suffix	Appliance for
.tgz	Deploying via PXE from a central server

10.2.4 Deployment

To help you deploy the OBS server appliance to a hard disk there is a basic installer that you can boot from a USB stick. The installer can be found on the OBS Download page (http://openbuildservice.org/download/).

The image can be written to a USB stick to boot from it:

```
xzcat obs-server-install.x86_64.raw.xz > /dev/sdX
```

Warning

/dev/sdX is the main device of your USB stick. Do NOT put it into a partition like /dev/sda1. If you use the wrong device, you will destroy all data on it!

How to deploy the other image types deeply depends on your virtualization setup. Describing this is out of scope for this guide, sorry.

10.2.5 Separating Data from the System

For production use you want to separate the OBS data from operating system of the appliance so you can re-deploy the appliance without touching your OBS data. This can be achieved by creating an LVM volume group with the name "OBS". This volume group should be as large as possible because it is getting used by the OBS back-end for data storage and the OBS workers for root/swap/cache file systems. To create an LVM volume prepare a partition of type "8e" and create the LVM via

```
pvcreate /dev/sdX1
vgcreate "OBS" /dev/sdX1
```

Additionally, if the OBS volume group contains a logical volume named “server”, it will be used as the data partition for the server.

```
lvcreate "OBS" -n "server"
```

```
mkfs.xfs /dev/OBS/server
```

10.2.6 Updating the Appliance

All images come pre-configured with the right set of repositories and can be updated via the system tools YaST or `zypper` at any time. Another way to update is to re-deploy the entire image.

Warning

If you re-deploy the entire image, keep in mind that you need to have your data directory (`/srv/obs`) on a separate storage (LVM volume, partition etc.) otherwise it will be deleted!

10.3 Back-end Administration

10.3.1 Services

You can control the different back-end components via `systemctl`. You can enable/disable the service during booting the system and start/stop/restart it in a running system. For more information, see man page (https://www.freedesktop.org/software/systemd/man/systemctl.html#Commands). For example, to restart the repository server, use:

```
systemctl restart obsrepserver.service
```

TABLE 10.2: SERVICE NAMES

Component	Service Name
Repository Server	obsrepserver.service
Source Server	obssrcserver.service
Scheduler	obsscheduler.service
Dispatcher	obsdispatcher.service
Publisher	obspublisher.service

Component	Service Name
Worker	obsworker.service
Source Services	obsservice.service
Download On Demand Updates	obsdodup.service
Delta Storage	obsdeltastore.service
Signer	obssigner.service
Warden	obswarden.service

10.3.2 Advanced Setups

It makes sense to run some of the different components of the OBS back-end on isolated hosts.

10.3.2.1 Distributed Workers

OBS workers can be very resource hungry. It all depends on the software that is being built, and how. Single builds deep down in the dependency chain can also trigger a sea of jobs. It makes sense to split off workers from all the other services so they do not have to fight for the same operating system/hardware resources. Here is an example on how to setup a remote OBS worker.

1. Install the worker package called obs-worker
2. Configure the OBS repository server address in the file /etc/sysconfig/obs-server. Change the variable OBS_REPO_SERVERS to the host name of the machine on which the repository server is running: OBS_REPO_SERVERS = "myreposerver.example:5252"
3. Start the worker

10.4 Front-end Administration

The Ruby on Rails application is run through the Apache web server with mod_passenger (https://www.phusionpassenger.com/). You can control it via `systemctl`

```
systemctl {start, stop, restart} apache2
```

10.4.1 Delayed Jobs

Another component of the OBS front-end are delayed jobs for asynchronously executing longer tasks in the background. You can control this service also via **`systemctl`**.

```
systemctl {start, stop, restart} obsapidelayed
```

10.4.2 Full Text Search

The full-text search for packages and projects is handled by Thinking Sphinx (http://freelancing-gods.com/thinking-sphinx/). The delayed job daemon will take care of starting this service. To control it after boot, use the **`rake`** tasks it provides.

```
rake ts:{start, stop, rebuild, index}
```

11 Scheduling and Dispatching

One of the major functionalities of OBS is to calculate always the current state, based on available sources, binaries and user configurations. In case a change happened it will trigger builds to achieve a clean state again. The calculation of the need of a build job is called scheduling here. The assignment of a build job to a concrete build host (aka worker) is called dispatching.

11.1 Definition of a Build Process

A build process is calculated and executed based on the following

- The sources of a package defined which dependencies are required at build time. eg. `BuildRequires` lines in spec files defined which other packages must get installed to build a package
- The project configuration of the package defines repositories and architectures to build for. In case other repositories are used as a base the configuration from there is also considered.
- Dependencies of packages which are required are considered as well.
- Constraints regarding the worker are considered. A package may require certain amount of resources or specific features to build. Check the constraints chapter for details. However, apart from this the build should be independent of the specific worker where the job gets dispatched to.

11.2 Scheduling Strategies

The defaults have the goal of creating an always reproducible state. This may lead to more builds than practically necessary, but ensures that no hidden incompatibilities exist between packages and also that the same state can later be achieved again (with a subsequent rebuild of the same sources and configurations). This can also lead to multiple builds of the same package in the case of dependency loops.

In some setups this may not be wanted, so each repository can be configured differently. The usual options to modify the project meta configurations can be used to configure different strategies. For example using osc:

```
osc meta prj -e YOUR_PROJECT
```

A repository is configured as following by default, however only the name attribute is required to be set.

```
# Example <repository
    name="standard" rebuild="transitive" block="all" linkedbuild="off"> [...]
    </repository>
```

11.2.1 Build Trigger Setting

The build trigger setting can be set via the "rebuild" attribute. Possible settings are

transitive

The default behavior, do a clean build of all dependant packages

direct

Just build the package with changed sources and direct dependant packages. But not indirect dependant packages.

local

Just build packages with changed sources.

Note

Note: You can run into dependency problems in case you select `direct` or `local` without noticing this in the build state. Your packages might not even be installable or have random runtime errors (like not starting up or crashing applications), even when they claim to be "succeeded". Also, you cannot be sure that you will be able to re-build them later. So never do an official shipment with this setting of a release. This knob is exposed to allow deliberate suppression of the strictly reproducible builds (for example, to limit burning CPU unnecessarily).

11.2.2 Block Mode

Usually the build of a package gets blocked when a package required to build it is still building at the moment. The "block" attribute can modify this behaviour:

all

The default behavior, do not start the build if a dependant package is currently building.

local

Just care about packages in your project for the block mode.

never

Never set a package to blocked.

Note

When using something other than “all” you will have to deal with the following problems:

- Intermediate builds can have dependency and runtime problems.
- Your packages will get built more often, take more resources on the server side. As a result the dispatcher will rate your repository down.

11.2.3 Follow Project Links

off

DEFAULT: do not build packages from project links

localdep

only build project linked packages if they depend on a local package

all

treat packages from project links like local packages

12 Build Job Constraints

Build job constraints can define requirements for the hardware or software of the build host. Constraints can be defined per package or for repositories.

The build constraints for an entire project or specific repositories is part of the project config. For each constraint, it contains a line

```
Constraint: <SELECTOR> <STRING>
```

The selector is a colon-separated list.

The build constraints for a package are part of the package sources, as a _constraints XML source file (validated on submission). The _constraints source file can contain the values listed below.

NOTE: If no build host meets the constraints, the package will stay in state "scheduled" and never be dispatched.

12.1 `hostlabel`

The hostlabel is any string which can be assigned to build hosts when starting the bs_worker process. It can be used to run on specific hosts, which may be used for running benchmarks in a reproducible way. This constraint can also be defined as a negative definition using the `exclude=true` attribute. However, the hostlabel is always specific to one OBS instance. You should avoid it as much as possible, since building with this constraint in another instance is usually not possible. Use any of the other constraints if possible.

Example for _constraints file:

```
<constraints exclude="false">
  <hostlabel>benchmark_runner</hostlabel>
</constraints>
```

Example for project configuration:

```
Constraint: hostlabel benchmark_runner
```

12.2 sandbox

Defines the sandbox which is used for the build job. Sandboxes are chroot, Xen or KVM environments. There is also the virtual secure sandbox, which allows building on Xen or KVM. This constraints may also be defined as a negative definition via the `exclude=true` attribute.

Example for `_constraints` file:

```
<constraints exclude="true">
  <sandbox>secure</sandbox>
</constraints>
```

Example for project configuration:

```
Constraint: sandbox secure
```

12.3 linux

The Linux kernel specific part.

12.3.1 version

To require a specific Linux kernel version.

Example for _constraints file:

```
<constraints>
  <linux><version>
    <min>3.0</min>
    <max>4.0</max>
  </version></linux>
</constraints>
```

Example for project configuration:

```
Constraint: linux:version:min 3.0
Constraint: linux:version:max 4.0
```

12.3.1.1 min

Minimal kernel version.

12.3.1.2 max

Maximal kernel version.

12.3.2 flavor

A specific kernel flavor like default or smp (from kernel packages kernel-default or kernel-smp).

Example for _constraints file:

```
<constraints>
  <linux>
    <flavor>default</flavor>
  </linux>
</constraints>
```

Example for project configuration:

```
Constraint: linux:flavor default
```

12.4 hardware

To require hardware or build instance features.

12.4.1 cpu

To require a specific Linux kernel version.

12.4.1.1 flag

CPU features which are provided by the hardware. On Linux they can be found in /proc/cpuinfo. The flag element may be used multiple times to require multiple CPU features.

Example for _constraints file:

```
<constraints>
  <hardware><cpu>
    <flag>mmx</flag>
    <flag>sse2</flag>
```

```
  </cpu></hardware>
</constraints>
```

Example for project configuration:

```
Constraint: hardware:cpu:flag mmx
Constraint: hardware:cpu:flag sse2
```

12.4.2 processors

To require a minimal number of processors for the build job.

Example for _constraints file:

```
<constraints>
  <hardware>
    <processors>4</processors>
  </hardware>
</constraints>
```

Example for project configuration:

```
Constraint: hardware:processors 4
```

12.4.3 disk

Hard disk specific.

12.4.3.1 size

To require a minimal size of the disk.

Example for _constraints file:

```
<constraints>
  <hardware>
    <disk>
      <size unit="G">4</size>
    </disk>
  </hardware>
</constraints>
```

Example for project configuration:

```
Constraint: hardware:disk:size unit="G" 4
```

12.4.4 memory

Memory specific.

12.4.4.1 size

To require a minimal memory size including swap space.

Example for _constraints file:

```
<constraints>
  <hardware>
    <memory>
      <size unit="M">1400</size>
    </memory>
  </hardware>
</constraints>
```

Example for project configuration:

```
Constraint: hardware:memory:size unit="M" 1400
```

12.4.5 physicalmemory

Memory specific.

12.4.5.1 size

To require a minimal memory size. Swap space is not taken into account here.

Example for _constraints file:

```
<constraints>
  <hardware>
    <physicalmemory>
      <size unit="M">1400</size>
```

```
    </physicalmemory>
  </hardware>
</constraints>
```

Example for project configuration:

```
Constraint: hardware:physicalmemory:size unit="M" 1400
```

12.5 Constraint Handling

The constraint handling depends on what is met by the restrictions. The handling starts when there is no worker to fulfill the constraints.

12.5.1 More than half of the workers satisfy the constraints

The job will just stay in state scheduled and no further notification is set.

12.5.2 Less than Half of the Workers Satisfy the Constraints

The job will stay in state scheduled and the dispatch details are set to tell the user that this job can take a long time to be built. This will be shown in the Web UI on mouse over and the scheduled state will be highlighted as well.

```
waiting for 4 compliant workers (4 down)
```

The **`(4 down)`** means that 4 of the 4 compliant workers are down and that someone should have a look.

12.5.3 No Workers Satisfy the Constraints

If no worker can handle the constraints defined by the package or project the build job fails. There is also a hint in the build log what has failed.

```
package build was not possible:

no compliant workers (constraints mismatch hint: hardware:processors sandbox)
```

```
Please adapt your constraints.
```

12.6 Checking Constraints with `osc`

You can check the constraints of a project / package with the osc tool. You have to be in an osc working directory.

```
osc checkconstraints [OPTS] REPOSITORY ARCH CONSTRAINTSFILE
```

Either you give a repository and an arch or osc will check the constraints for all repository / arch pairs for the package. A few examples:

```
# osc checkconstraints
Repository                Arch                      Worker
----------                ----                      ------
openSUSE_Leap_42.2        x86_64                    1
openSUSE_Leap_42.1        x86_64                    1
```

If no file is given it takes the local _constraints file. If this file does not exist or the --ignore-file switch is set only the project constraints are used.

```
# osc checkconstraints openSUSE_Leap_42.1 x86_64
Worker
------
x86_64:worker:1
x86_64:worker:2
```

If a repository and an arch is given a list of compliant workers is returned.

Another command to verify a worker and display the worker information is osc workerinfo.

```
<worker hostarch="x86_64" registerserver="http://localhost:5252" workerid="worker:1">
  <hostlabel>MY_WORKER_LABEL_1</hostlabel>
  <sandbox>chroot</sandbox>
  <linux>
    <version>4.1.34-33</version>
    <flavor>default</flavor>
  </linux>
  <hardware>
    <cpu>
      <flag>fpu</flag>
      <flag>vme</flag>
      <flag>de</flag>
    </cpu>
```

```
    <processors>2</processors>
    <jobs>1</jobs>
  </hardware>
</worker>
```

It returns the information of the desired worker.

13 Building Preinstall Images

Preinstall images can optionally be used to install a set of packages in one quick step instead via single package installations. Depending on the build host even snapshots with copy-on-write support may be used which avoids any IO.

A preinstall image can be used if it provides a subset of packages which is required for the build job. The largest possible image is taken if multiple are usable.

To use a preinstall image there needs to be a package container inside of the project or in a repository used by the build job. This package needs a _preinstallimage file. The syntax of it is spec file like, but just needs a Name: and at least one BuildRequires: line.

To ignore packages despite existing dependencies, use `#!BuildIgnore:` tags or `%if`.

Preinstall image build jobs are always preferred to allow the best effect of them. We recommend defining images for often used standard stacks.

Example _preinstallimage file for a basic preinstall image:

```
Name: base
BuildRequires: bash
#!BuildIgnore: brp-trim-desktopfiles
```

14 Authorization

14.1 OBS Authorization Methods

Each package is signed with a PGP key to allow checking its integrity on user's machines.

14.1.1 Default Mode

OBS provides its own user database which can also store a password. The authentication to the API happens via HTTP BASIC AUTH. See the API documentation to find out how to create, modify or delete user data. Also a call for changing the password exists.

14.1.2 Proxy Mode

The proxy mode can be used for esp. secured instances, where the OBS web server shall not get connected to the network directly. There are authentication proxy products out there which do the authentication and send the user name via an HTTP header to OBS. This also has the advantage that the user password never reaches OBS.

14.1.3 LDAP Mode

LDAP authentication code is still part of OBS, but due to the lack of any test cases it is currently not recommended to use it.

14.2 OBS Token Authorization

OBS provides a mechanism to create tokens for specific operations. This can be used to allow certain operations in the name of a user to others. This is esp. useful when integrating external infrastructure. The create token should be kept secret by default, but it can also be revoked at any time if it became obsolete or leaked.

14.2.1 Managing User Tokens

Tokens always belong to a user. A list of active tokens can be viewed using

```
osc token
```

```
osc token --delete <TOKEN>
```

14.2.2 Executing a Source Service

A token can be used to execute a source service. The source service has to be setup for the package first, check the source service chapter for this. A typical example is to update sources of a package from git. A source service for that can be setup with:

```
osc add git://....
```

The best way to create a token is bind it to a specific package. The advantage is that the operation is limited to that package, so less bad things can happen when the token leaks.

```
osc token --create <PROJECT> <PACKAGE>
```

Also, you do not need to specify the package at execution time. But keep in mind that such form only works when you run it on an as checkout of a package. Both commands below do the same thing but in a different way:

```
osc token --trigger <TOKEN>
```

```
osc api -X POST /trigger/runservice?token=<TOKEN>
```

A token can be registered as generic token, means allowing to execute all source services in OBS if the user has permissions. You can create such a token by skipping project/package on creation command:

```
osc token --create
```

In this case, you are forced to specify project/package along with the token. On the other hand, you are not limited from where you execute it. Again, two examples doing same thing:

```
osc token --trigger <TOKEN> <PROJECT> <PACKAGE>
```

```
curl -H "Authorization: Token <TOKEN>" -X POST /trigger/runservice?
project=<PROJECT>&package=<PACKAGE>
```

You can also limit the token to a specific package. The advantage is that the operation is limited to that package, so less bad things can happen when the token leaks. Also you do not need to specify the package on execution time. Create and execute it with:

```
osc token --create <PROJECT> <PACKAGE>
```

```
osc token --trigger <TOKEN>
```

```
curl -H "Authorization: Token <TOKEN>" -X POST /trigger/runservice
```

15 Quality Assurance(QA) Hooks

OBS provides multiple hooks to place automated or manual tests at different points of time.

This chapter describes the different possibilities to provide and execute QA checks. The order of the items is sorted by the order in a typical development process. It is preferred to add a check as early as possible in the process to keep turn-around times small.

15.1 Source Related Checks

Things which can be verified based on a given source can be checked even before commit time on the developers workstation. This is the earliest possible point of time to add a check. But it can also optionally be enforced on the server side.

Automated source processing is done by source services in OBS world. Check the source service chapter how to use or write one. It is important to decide if the test case shall output warning messages and when it shall report an error by exit status.

Test cases in source services get usually applied to all packages of a project. (It is possible to execute it only for specific packages though.)

15.2 Build Time Checks

15.2.1 In-Package Checks

Checks running during the build of a package are usually test cases provided by the author of a package. However, the packager can also add simple checks, for example, for code that is known to break on version updates and might be forgotten when the package is touched the next time.

These test are often specific for a concrete package only. So it is typically executed in %check section of rpm spec files directly. In case the check can be used with multiple package source, it is a good idea to package the test case in an own package and just call it from the other packages.

rpm calls %check after %install section and before creating the actual checks.

SUSE distributions also provide build time checks to test the installed files inside of the build root. It is to be used for test cases which shall run on all packages which are build inside of a distribution. This hook can be used by installing a file to /usr/lib/rpm/brp-suse.d/ directory. These scripts also have the power to modify installed files if needed.

15.2.2 Post Build Checks

The standard tool to test binary packages for RPM-based distributions is **rpmlint**. DEB-based distributions use the **lintian** tool instead.

These checks are executed by the build script after a successful build. Note that these are executed as the standard user by default.

15.2.3 Post Build Root Checks

Files in /usr/lib/build/checks/* are executed as root user. Typical use cases are install tests of the build packages to ensure that the scripts inside of the packages are working in general.

15.2.4 KIWI Specific Post Build Root Checks

The file /usr/lib/build/kiwi_post_run is executed after KIWI jobs have finished. It can be used to run the appliance or to modify it. For example to package an appliance into an rpm.

15.3 Workflow Checks

Workflow steps, for example transferring packages from one project to another, are done via requests in OBS. At least when multiple parties are involved. One or more of these parties can be automated test cases. Or human manual approval steps.

Default reviews can be defined inside of projects and packages. A new request to a certain package does get the reviewers added defined in target projects and packages. Reviewers can be currently users, groups or the maintainers of a specified project or package.

15.3.1 Automated Test Cases

Open requests can be requested in an XML parseable way via the API running

```
osc api /request?states=review&user=auto-review-
user&roles=reviewer&reviewstates=new&view=collection
```

`osc` can be used to accept or decline requests after running the automated test. It can also add a review comment which you can use to give a reason (for example, error messages) for accepting or declining a request. Requests, which are not tested, for example because they are of a not matching type (for example, deleting packages) needs to get also a review accept. Otherwise, this would block the process.

16 openSUSE Factory

This chapter describes how the development of the future openSUSE distribution is done within OBS.

16.1 openSUSE:Factory project

The main project is openSUSE:Factory. This project is controlled by a small group which does review all submissions according to the policies. Submissions are possible via submit requests, which are reviewed by default by two groups: The Legal team and the code review team.

16.2 Devel Projects

The goal of openSUSE:Factory is to always have a working state. This is needed to allow all developer groups to use it as a base for testing their own, possibly experimental work in their own projects.

Glossary

AppImage

An application and its dependencies packaged as a single file which can run on many distributions without unpacking or installing.

Appliance

An image built and preconfigured for a specific purpose. Appliances usually consist of a combination of an application (for example, a Web server), its configuration, and an operating system (for example, SUSE Linux Enterprise Server). Appliances can be copied as-is onto a hard disk, an SSD, or started as a virtual machine (*deployed*).

See also Operating System Image, Image (Image File).

Archive (Archive File)

An archive file contains a representation of usually multiple files and directories. Usually, archive files are also compressed. Archive files are the basis for binary packages (*Binary Package (Binary)*).

Attribute

Attributes can be added to projects or packages to add meta information or to trigger actions. For example, you can use the attribute `OBS:AutoCleanup` to delete a project after a certain amount of time.

Binary Package (Binary)

An archive file that contains an installable version of software and metadata. The metadata includes references to the dependencies of the main software. Dependencies are packaged as additional binary packages.

Formats of binary packages include RPM and DEB. In the OBS context, binary packages are sometimes also called *binaries*.

See also Container, Operating System Image, Source Package, Deb, RPM, KIWI, Archive (Archive File).

Branch

Personal copy of another repository that lives on your home project. A branch allows you to make changes without affecting the original repository. You can either delete the branch or merge it into the original repository with a submit request.

See also Submit Request.

Bug

Issue that documents incorrect or undesirable behaviour

Bugowner

In OBS, *Bugowner* is a user role which can be set for a project or a package. However, ideally, set this role for individual packages only. Users with this role can only read data but they are responsible for reacting to bug reports.

See also Maintainer.

Build

Generating ready-to-publish binaries, usually for a specific distribution and architecture.

Build Log

Output of the build process of a certain package.

See also Build.

Build Recipe

Generic term for a recipe file for creating a package. A build recipe includes metadata, instructions, requirements, and changelogs. For RPM-based systems like SUSE, a `.spec` file is used and contains all the previous points. Debian-based systems use a `debian` directory which splits all the information.

See also Spec File.

Build Requirement

Package requirements that are needed to create or build a specific package.

See also Installation Requirement, Build Recipe.

Build Result

The current state of a package. Example of a build result could be succeeded, failed, blocked, etc.

Build Root

Directory where the **`osc`** command copies, patches, builds, and create packages. By default, the build root is located in `/var/tmp/build-root/`*`BUILD_TARGET`*.

See also Build Target.

Build Target

Specific operating systems and architecture to build for.

Changelog

Listing of a high-level overview sorted by date. An entry of a changelog can contain information about version updates, bug and security fixes, incompatible changes, or changes related to the distribution.

See also `.changes` File.

`.changes` File

In OBS, a file with the file extension `.changes` to store changelog information.

See also Changelog.

Commit

A record of a change to one or more files. Each record contains the revision, the author, the date and time, a commit checksum, an optional request number, and a log message.

See also Revision.

Container

An image file that contains a deployable version of software and metadata. Dependencies of the main software are also included, such as additional libraries.

Unlike operating system images, containers do not include an operating system. Unlike binary packages, containers are deployed and not installed. Formats of containers include AppImage, Docker, Snap, and Flatpak.

See also Binary Package (Binary), Operating System Image, Image (Image File).

Deb

A package format created and used by the Debian distribution.

See also Package, RPM.

Dependency

See Requirement.

Devel Project

A set of related packages that share certain features. For example, the devel project `devel:languages:python` stores all packages related to the Python programming language.

See also Home Project, Project.

Docker

Docker is a lightweight virtualization solution to run multiple virtual units (containers) simultaneously on a single control host.

See also Container.

Download Repository

An area containing built packages available for download and installation through Zypper or YaST. The download repository belongs to a project and is specific to a distribution. An example of a download repository could be `http://download.opensuse.org/repositories/PROJECT/openSUSE_Tumbleweed/`.

Diff

See Patch.

EULA

End User License Agreement. For software that needs a special license (usually non-open source) which the user needs to agree to before installing.

Fix

See Patch.

Flags

A set of switches that determine the state of package or repository. This includes building, publishing, and generating debug information.

GA Project

The GA (general availability) project builds an initial release of a product. It gets frozen after releasing the product. All further updates get released via the *Update Project* of this project.

GPG Key

An encryption key pair that in the context of OBS is used to verify the owner of the repository and packages.

Home Project

Working area in OBS for uploading and building packages. Each home project starts with `home:USERNAME`.

See also Project.

Image (Image File)

An image file contains a bit-wise representation of the layout of a block device. Some types of image files are compressed. OBS allows building multiple types of image:

Operating System Image, *Container*

Image Description

Specification to define an appliance built by KIWI. The image description is a collection of files directly used by KIWI (`config.xml` and `*.kiwi`), scripts, or configuration data to customize certain parts of the KIWI build process.

See also KIWI.

Incident

Describes a specific problem and the required updates. If the problem exists for multiple code streams, one incident covers all of them. An incident is started by creating a maintenance incident project and the update get built here.

Installation Requirement

Package requirements that are needed when the package is installed.

KIWI

A tool to build operating system images. It can create images for Linux supported hardware platforms or for virtualization systems.

See also Image (Image File).

License

Written contract to specify permissions for use and distribution of software.

See also Project.

Link

A concept that defines a relationship between a source and a target repository.

See also Project.

Maintainer

In OBS, *Maintainer* is a user role which can be set for a project or a package. Users that have this role in a project can add, modify, and remove packages and subprojects, accept submit requests, and change metadata.

See also Bugowner.

Maintenance Project

A project without sources and binaries, defined by the maintenance team. Incidents are created as sub projects of this project.

See also Incident.

OBS Package

OBS packages contain the sources that are necessary to build one or more binary packages or containers. The content of OBS packages varies. In general, there is always a source file (such as a TAR archive of the upstream sources) and a build recipe.

To build an RPM package in OBS, you need a spec file as your build recipe, for example. An OBS package can also contain other files, such as a change log and patches.

OBS packages, unlike the name "package" suggests, do not consist of a single file. Instead, they are directories of a version-controlled repository. However, unlike most directories, they cannot contain subdirectories. (You can use subdirectories to simplify your work with the checked-out package but you cannot submit these directories.)

Open Build Service (OBS)

A Web service to build binary packages, containers and operating system images from source.

The term "Open Build Service" is used to speak about the server part of the build service. Unlike the term openSUSE Build Service, the term Open Build Service refers to all instances.

openSUSE Build Service

A specific Web service instance of *Open Build Service (OBS)* from the openSUSE project at http://build.opensuse.org.

`osc`

A command line tool to work with OBS instances. The acronym `osc` stands for *openSUSE commander*. `osc` works similarly to SVN or Git.

See also Open Build Service (OBS), https://github.com/openSUSE/osc.

Operating System Image

An image file that contains an operating system. The operating system can be either installable or deployable. Depending on their purpose, operating system images are classified into: *Product Image*, *Appliance*, *Virtual Machine Image*

Formats of operating system images include ISO, Virtual Disk, and PXE Root File System.

See also Binary Package (Binary), Image (Image File), KIWI.

Overlay File

A directory structure with files and subdirectories used by KIWI. This directory structure is packaged as a file (`root.tar.gz`) or stored below a directory (named `root`). The contents of the directory structure is copied over the existing file system (overlaid) of the appliance root. This includes permissions and attributes as a supplement.

See also Appliance, KIWI.

Package

OBS handles very different types of software package:
Source Package, *OBS Package*, *Binary Package (Binary)*
See also Container.

Package Requirement

See Requirement.

Package Repository

A place where installable packages are available. This can be either from a media like CD, DVD, or from a remote online repository.

Official repositories can divided into oss software (licensed under an open source license) and non-oss (for software released under other. non-open source licenses). Additionally, there are update source, and debug repositories as well.

Patch

Textual differences between two versions of a file.
See also Patch File.

Patch File

A file that contains a patch with the file extension `.diff` or `.patch`.
See also Patch.

Product Image

An image that allows installing an operating system, usually from a removable medium, such as a USB disk or a DVD onto a hard disk or SSD.

Live images are a special case of operating system images. They can be run directly a USB disk or DVD and are often but not always installable.
See also Operating System Image, Image (Image File).

Project

Unit which definies access control, repositories, architectures, and a set of packages containing sources.

Project Configuration

Settings to define the setup of the build system, usually to switch on or off certain features during the build or to handle circular dependencies.

See also Project.

Publishing

Finished process when a package is successfully build and available in the download repository.

See also Download Repository.

Release Project

A release project is hosting a release repository which is not building any packages ever. It is only used to copy sources and binaries to this project on a release event.

Repository

A distribution-specific area that holds dependencies required for building a package.

See also Download Repository.

Repo File

A file with the name *`PROJECT`*`.repo`. inside the download repository. The file contains information about the name of the repository, the repository type, and references to the download repository and the GPG key.

See also Download Repository.

Requirement

In the OBS context, package requirements that are needed to create, build, or install a package.

See also Build Requirement, Installation Requirement.

Revision

A unique numeric identifier of a commit.

See also Commit.

RPM

A package format. It stands for recursive acronym RPM Package Manager. Mainly used by SUSE, Red Hat, u.a.

See also Deb, Package.

Sandbox

Isolated region of a host system which runs either a virtual machine or a change root environment.

See also Build Root.

Service File

An XML file that contains metadata required for building a package. This includes version information, upstream source repository, and actions.

Spec File

A file that contains metadata and build instructions. Metadata includes a general package description and dependencies required for building and installing the package.

See also Build Recipe, Patch, Source.

Source

Original form, mostly written in a computer language.

See also Package.

Source Link

See Link.

Source Package

Source packages contain content similar to an OBS package but they are packaged in an archive file. They are also meant to allow building a single binary package or container format only. However, source packages allow rebuilding outside of an Open Build Service context.

An example of source packages are SRPMs which contain the source for accompanying RPM binary packages.

See also Binary Package (Binary), Archive (Archive File).

Source Service

A tool to validate, generate, or modify a source in a trustable way.

See also Source.

SUSE Package Hub

An OBS project reachable under `openSUSE:Backports`. It is a subset of openSUSE Factory which does not contain version updates and does not conflict with official packages supported by SUSE Linux Enterprise.

Submit Request

Asking for integrating changes from a branched project.

Subproject

A child of a parent project.

See also Devel Project, Home Project, Project.

Target

A specific distribution and architecture, for example, openSUSE Tumbleweed for x86-64. Also referenced as *build target*.

Update Project

A project which provides official updates for the products generated in the *GA Project*. The update project usually links sources and repositories against the *GA Project*.

See also Release Project, GA Project.

Virtual Machine Image

An image which is built (and sometimes preconfigured) to be the basis of virtual machines. Such images can usually be copied to the target computer and run as-is. As such, there is some overlap between virtual machine images and appliances.

See also Operating System Image, Image (Image File).

Watchlist

A list of repositories that the user is interested in, available in the OBS Web UI.

Working Copy

See Working Directory.

Working Directory

A directory on your local machine as a result from a **`osc checkout`** call for working and building before submitting your changes to an OBS instance.

Zypper

A command line package manager to access repositories, solve dependencies, install packages, and more.

A GNU Licenses

This appendix contains the GNU General Public License version 2 and the GNU Free Documentation License version 1.2.

GNU General Public License

Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc. 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a). You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b). You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c). If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a). Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b). Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c). Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER

EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the “copyright” line and a pointer to where the full notice is found.

```
  one line to give the program’s name and
an idea of what it does.
  Copyright (C) yyyy name of author
```

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

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

```
  You should have received a copy of the
GNU General Public License
  along with this program; if not, write
to the Free Software
  Foundation, Inc., 59 Temple Place -
Suite 330, Boston, MA 02111-1307, USA.
```

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

```
  Gnomovision version 69, Copyright (C)
year name of author
  Gnomovision comes with ABSOLUTELY NO
WARRANTY; for details
  type `show w’. This is free software,
and you are welcome
  to redistribute it under certain
conditions; type `show c’
  for details.
```

The hypothetical commands `show w’ and `show c’ should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than `show w’ and `show c’; they could even be mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a “copyright disclaimer” for the program, if necessary. Here is a sample; alter the names:

```
  Yoyodyne, Inc., hereby disclaims all
copyright
  interest in the program `Gnomovision’
  (which makes passes at compilers)
written
  by James Hacker.
```

```
  signature of Ty Coon, 1 April 1989
  Ty Coon, President of Vice
```

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License (http://www.fsf.org/licenses/lgpl.html)↗ instead of this License.

GNU Free Documentation License

Version 1.2, November 2002

Copyright (C) 2000,2001,2002 Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document “free” in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of “copyleft”, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The “Document”, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as “you”. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A “Modified Version” of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A “Secondary Section” is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The “Invariant Sections” are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The “Cover Texts” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A “Transparent” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not “Transparent” is called “Opaque”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The “Title Page” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

A section “Entitled XYZ” means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as “Acknowledgements”, “Dedications”, “Endorsements”, or “History”.) To “Preserve the Title” of such a section when you modify the Document means that it remains a section “Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document’s license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License “or any later version” applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
  Copyright (c) YEAR YOUR NAME.
  Permission is granted to copy,
 distribute and/or modify this document
  under the terms of the GNU Free
 Documentation License, Version 1.2
  or any later version published by the
 Free Software Foundation;
  with no Invariant Sections, no Front-
Cover Texts, and no Back-Cover Texts.
  A copy of the license is included in
 the section entitled “GNU
  Free Documentation License”.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the “with...Texts.” line with this:

```
  with the Invariant Sections being LIST
 THEIR TITLES, with the
  Front-Cover Texts being LIST, and with
 the Back-Cover Texts being LIST.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.