OpenDaylight Documentation

Release Chlorine

OpenDaylight Project

Oct 09, 2023

CONTENTS

The OpenDaylight documentation site acts as a central clearinghouse for OpenDaylight project and release documentation. If you would like to contribute to documentation, refer to the *Documentation Guide*.

CHAPTER

ONE

GETTING STARTED WITH OPENDAYLIGHT

1.1 OpenDaylight Downloads

1.1.1 Supported Releases

Argon-SR2

(Current Release)

Announcement
Argon

GA Release Date
April 2, 2023

Service Release Date
September 6, 2023

Downloads

- OpenDaylight Argon Tar
- OpenDaylight Argon Zip

Documentation

- Getting Started Guide
- Project Guides
- Release Notes

Chlorine-SR3

Announcement
Chlorine

GA Release Date
October 27, 2022

Service Release Date
June 26, 2023

Downloads

- OpenDaylight Chlorine Tar

- OpenDaylight Chlorine Zip

Documentation

- Getting Started Guide
- Project Guides
- Release Notes

1.1.2 Docker images

- OpenDaylight Docker Hub

1.1.3 Archived Releases

- OpenDaylight (Fluorine and newer)
- OpenDaylight (Nitrogen and Oxygen)
- OpenDaylight (Carbon and earlier)
- ODL Micro
- NeXt UI
- VTN Coordinator
- OpFlex

1.2 Release Notes

1.2.1 Execution

OpenDaylight includes Karaf containers, OSGi (Open Service Gateway Initiative) bundles, and Java class files, which are portable and can run on any Java 17-compliant JVM (Java virtual machine). Any add-on project or feature of a specific project may have additional requirements.

1.2.2 Development

OpenDaylight is written in Java and utilizes Maven as a build tool. Therefore, the only requirements needed to develop projects within OpenDaylight include:

- Java JDK 17
- Apache Maven 3.8.3 or later

If an application or tool is built on top of OpenDaylight's REST APIs, it does not have any special requirement beyond what is necessary to run the application or tool to make REST calls.

In some instances, OpenDaylight uses the Xtend language. Even though Maven downloads all appropriate tools to build applications; additional plugins may be required to support IDE.

Projects with additional requirements for execution typically have similar or additional requirements for development. See the platforms release notes for details.

Platform Release Notes

2022.09 Chlorine Platform Upgrade

This document describes the steps to help users upgrade from Sulfur to Chlorine planned platform. Refer to Managed Snapshot Integrated (MSI) project upgrade patches for more information and hints for solutions to common problems not explicitly listed here.

Contents

Preparation

JDK 17 Version

2022.09 Chlorine requires Java 17, both during compile-time and run-time. Make sure to install JDK 17 corresponding to at least `openjdk-17.0.4`, and that the JAVA_HOME environment variable points to the JDK directory.

Version Bump

Before performing platform upgrade, do the following to bump the odlparent versions (for example, bump-odl-version):

1. Update the odlparent version from 10.0.3 to 11.0.1. There should not be any reference to **org.opendaylight.odlparent**, except for 11.0.1. This includes custom feature.xml templates (`src/main/feature/feature.xml`), the version range should be "[11,12)" instead of "[10,11)", "[5.0.3,6)" or any other variation.

```
bump-odl-version odlparent 10.0.3 11.0.1
```

2. Update the direct yangtools version references from 8.0.7 to 9.0.1, There should not be any reference to **org.opendaylight.yangtools**, except for 9.0.1. This includes custom feature.xml templates (`src/main/feature/feature.xml`), the version range should be "[9,10)" instead of "[8,9)".

```
bump-odl-version yangtools 8.0.7 9.0.1
```

3. Update the MD-SAL version from 9.0.5 to 10.0.2. There should not be any reference to **org.opendaylight.mdsal**, except for 10.0.2.

```
bump-odl-version mdsal 9.0.5 10.0.2
```

4. Update the Controller version from 5.0.6 to 6.0.2. There should not be any reference to **org.opendaylight.controller**, except for 6.0.2.

```
bump-odl-version controller 5.0.6 6.0.2
```

5. Update the InfraUtils version from 3.0.2 to 4.0.1. There should not be any reference to **org.opendaylight.infrautils**, except for 4.0.1.

```
bump-odl-version infrautils 3.0.2 4.0.1
```

6. Update the AAA version from 0.15.6 to 0.16.3. There should not be any reference to **org.opendaylight.aaa**, except for 0.16.3.

```
bump-odl-version aaa 0.15.6 0.16.3
```

7. Update the NETCONF version from 3.0.6 to 4.0.2. There should not be any reference to **org.opendaylight.netconf**, except for 4.0.2.

```
bump-odl-version netconf 3.0.6 4.0.2
```

Install Dependent Projects

Before performing platform upgrade, users must also install any dependent project. To locally install a dependent project, pull and install the respective sulfur-mri changes for any dependent project.

Perform the following steps to save time when locally installing any dependent project:

- For quick install:

```
mvn -Pq clean install
```

- If previously installed, go offline and/or use the no-snapshot-update option.

```
mvn -Pq -o -nsu clean install
```

Upgrade the ODL Parent

The following sub-section describes how to upgrade to the ODL Parent version 9. Refer to the ODL Parent Release Notes for more information.

Features

Any version range referencing version 10 of ODL Parent must be changed to "[11,12)" for ODL Parent 10.

```
<feature name="odl-infrautils-caches">
    <feature version="[11,12)">odl-guava</feature>
</feature>
```

ODL Parent Impacts

Upstream declarations removed

A number of declarations of upstream projects, which are no longer used in OpenDaylight, have been removed. This includes `Google Truth`, `commons-codec`, `commons-fileupload`, `commons-net`, `jsonassert`, `jungg` and `spring-osgi-mock`.

Partial migration to Jakarta

A number of Jakarta EE artifacts have been migrated from their legacy `javax` namespace to the new `jakarta` namespace. This does not affect Java packages, only dependency declarations.

YANG Tools Impacts

SemVer-based YANG parser import resolution removed

The ability to recognize OpenConfig semantic versions in `import` statements and use them to resolve the import to a matching module has been removed.

Multiple constructs are now `sealed`

A number of interfaces and classes are now sealed. This includes `ItemOrder`, `AbstractQName`, `ArgumentDefinition`, `YangExpr`, `ModelStatement`, `YangInstanceIdentifier`, `LeafSetNode` and `MapNode`. This improves clarity of their design, making them easier to use and infer about, but also makes it impossible to use Mockito to mock them. Users may need to use real implementations instead of mocks.

Decimal64 are required to match `fraction-digits`

When a `leaf` or `leaf-list` item has `type decimal64`, JSON and XML codecs will reject values which cannot be scaled to the matching `fraction-digits`.

MD-SAL Impacts

Improvements to generated `toString()` methods

This release changes how generated `toString()` methods work in TypeObjects and with respect to `byte[]` properties. Property names now do not include a leading underscore. Byte array properties are now hex-encoded.

Mapping of `identityref` types changed

The Binding mapping of `type identityref` properties has changed. Given the following YANG snippet:

```
identity foo;

leaf bar {
  type identityref {
    base foo;
  }
}
```

We see an interface `Foo` generated for the identity. This remains unchanged, but when setting the `bar` leaf, rather than using `Foo.class`, users now need to specify `Foo.VALUE`. This also affects use of `type identityref` inside a `type union`: each such use now gets its own property.

Builders no longer generated for union types

Due to historic reasons, code generated for `type union` statements included a Builder, which was generated in the `src/main/java` directory hierarchy. This Builder was hosting only a single `getDefaultInstance()` method, which needed to be hand-coded.

All of this mechanics has been removed and users are advised to remove these hand-crafted classes.

Controller Impacts

No impacts in this release.

Project Release Notes

AAA

Overview

AAA (Authentication, Authorization, and Accounting) are services that help improve the security posture of an OpenDaylight deployment. By default, the majority of OpenDaylight's northbound APIs (and all RESTCONF APIs) are protected by AAA after installing the +odl-restconf+ feature.

Behavior/Feature Changes

There are no changes to features.

New Features

This release contains a major upgrade of H2 database. This impacts the ability to perform in-place upgrades. Users perfoming an upgrade will need to remove `data/idmlight.db.*` files and re-populate the database.

Deprecated and Removed Features

There are no deprecated or removed features.

Resolved Issues

The following table lists the issues resolved in this release.

Table 1: Issues resolved in versions 0.16.0 through 0.16.3 (JIRA)

Type	Key	Summary	Resolution	Fix Version(s)
	AAA-197	[CSRF] Attacker can insert or modify the entry of flow table	Duplicate	0.16.0
	AAA-229	ODLAuthenticator does not work	Duplicate	0.16.1
	AAA-230	web-impl-osgi mis-represents servlet paths	Done	0.16.2
	AAA-231	Resources not found with web-osgi-impl	Done	0.16.2
	AAA-232	WebInitializer failure with web-jetty-impl	Done	0.16.2
	AAA-235	StackOverflowError in aaa-filterchain	Done	0.16.3
	AAA-215	Shiro throws a warning about SecurityManager	Done	0.14.14, 0.15.6, 0.16.1
	AAA-213	Remove CORS filter from shiro-impl	Done	0.16.0
	AAA-225	Reimplement web-osgi-impl with HTTP Whiteboard	Done	0.16.0
	AAA-227	Bump Shiro to 1.9.1	Done	0.14.14, 0.15.6, 0.16.0
	AAA-221	Upgrade H2 database to 2.1.210	Done	0.16.0

Known Issues

The following table lists the known issues that exist in this release.

Table 2: Issues affecting versions 0.16.0 through 0.16.3 (JIRA)

Type	Key	Summary	Status	Affected Version(s)	Fix Version(s)
	AAA-240	SQL injection in the aaa-idm-store-h2 (deleteDomain function)	Resolved	0.15.0, 0.15.6, 0.16.0, 0.16.4	0.15.8, 0.16.5, 0.17.0
	AAA-241	SQL injection in the aaa-idm-store-h2 (deleteUser function)	Resolved	0.15.0, 0.15.6, 0.16.0, 0.16.4	0.15.8, 0.16.5, 0.17.0
	AAA-239	SQL injection in the aaa-idm-store-h2 (deleteRole function)	Resolved	0.15.0, 0.15.6, 0.16.0, 0.16.4	0.15.8, 0.16.5, 0.17.0

Resolved Issues in SR1

The following table lists the issues resolved in Service Release 1.

Table 3: Issues resolved in versions 0.16.4 through 0.16.6 (JIRA)

Type	Key	Summary	Resolution	Fix Version(s)
	AAA-240	SQL injection in the aaa-idm-store-h2 (deleteDomain function)	Done	0.15.8, 0.16.5, 0.17.0
	AAA-241	SQL injection in the aaa-idm-store-h2 (deleteUser function)	Done	0.15.8, 0.16.5, 0.17.0
	AAA-239	SQL injection in the aaa-idm-store-h2 (deleteRole function)	Done	0.15.8, 0.16.5, 0.17.0
	AAA-238	https configuration fails with blueprint errors	Done	0.15.6, 0.16.6, 0.17.0
	AAA-242	Upgrade Shiro to 1.10.1	Done	0.15.8, 0.16.5, 0.17.0

Known Issues in SR1

The following table lists the known issues that exist in Service Release 1.

Resolved Issues in SR2

The following table lists the issues resolved in Service Release 2.

Known Issues in SR2

The following table lists the known issues that exist in Service Release 2.

Resolved Issues in SR3

The following table lists the issues resolved in Service Release 3.

Table 4: Issues resolved in versions 0.16.8 through 0.16.9 (JIRA)

Type	Key	Summary	Resolution	Fix Version(s)
	AAA-247	Upgrade Shiro to 1.11.0	Done	0.15.9, 0.16.8, 0.17.3
	AAA-259	Align aaa's documentation version with distro	Done	0.16.9, 0.17.9, 0.18.0
	AAA-249	Fix AAA documentation	Done	0.16.8, 0.17.6

Known Issues in SR3

The following table lists the known issues that exist in Service Release 3.

BGP-PCEP

Overview

BGP Plugin

The OpenDaylight controller provides an implementation of BGP (Border Gateway Protocol), which is based on RFC 4271) as a south-bound protocol plugin. The implementation renders all basic *BGP speaker capabilities*, including:

Inter/Intra-AS peering * Routes advertising * Routes originating * Routes storage

The plugin's **north-bound API** (`REST/Java`) provides to user:

- Fully dynamic run-time standardized BGP configuration
- Read-only access to all RIBs
- Read-write programmable RIBs
- Read-only reachability/link-state topology view

PCEP Plugin

The OpenDaylight Path Computation Element Communication Protocol (PCEP) plugin provides all basic service units necessary to build-up a PCE-based controller. Defined by RFC 8231, PCEP offers LSP management functionality for Active Stateful PCE, which is the cornerstone for majority of PCE-enabled SDN solutions. It consists of the following components:

- Protocol library
- PCEP session handling
- Stateful PCE LSP-DB
- Active Stateful PCE LSP Operations

Behavior/Feature Changes

The configuration knob for `pcep-provider`'s timer has been moved into pcep-topology configuration. Users relying on non-default value need to update their configuration.

New Features

The Graph functionality has been extended to handle multiple PCEP topologies.

Deprecated Features

No deprecated features.

Resolved Issues

The following table lists the issues resolved in this release.

Table 5: Issues resolved in versions 0.18.0 through 0.18.2 (JIRA)

Type	Key	Summary	Resolution	Fix Version(s)
	BGPCEP-1010	Graph is not able to handle both IPv4 and IPv6 addresses	Done	0.18.0
	BGPCEP-1009	BGPDocumentedException with IPv6 prefix in BGP Link State	Done	0.16.16, 0.17.6, 0.18.0
	BGPCEP-1005	Race between topology manager and PCEP session	Done	0.18.0
	BGPCEP-1011	Allow statistics updates to be configured on a per-topology basis	Done	0.18.0
	BGPCEP-990	Update pcep-topology-type-config placement	Done	0.18.0
	BGPCEP-964	Remove blueprint from pcep-topology-stats	Duplicate	0.18.0

Known Issues

The following table lists the known issues that exist in this release.

Table 6: Issues affecting versions 0.18.0 through 0.18.2 (JIRA)

Type	Key	Summary	Status	Affected Version(s)	Fix Version(s)
	BGPCEP-1020	PCEP accesses dead transaction chain	Resolved	0.18.0, 0.18.4, 0.19.0, 0.19.3	0.18.5, 0.19.4

Resolved Issues in SR1

The following table lists the issues resolved in Service Release 1.

Table 7: Issues resolved in versions 0.18.3 through 0.18.4 (JIRA)

Type	Key	Summary	Resolution	Fix Version(s)
	BGPCEP-1015	PCUpd with empty LSP is sent	Done	0.17.8, 0.18.4, 0.19.0

Known Issues in SR1

The following table lists the known issues that exist in Service Release 1.

Table 8: Issues affecting versions 0.18.3 through 0.18.4 (JIRA)

Type	Key	Summary	Status	Affected Version(s)	Fix Version(s)
	BGPCEP-1020	PCEP accesses dead transaction chain	Resolved	0.18.0, 0.18.4, 0.19.0, 0.19.3	0.18.5, 0.19.4
	BGPCEP-1021	Failure to update OpenConfig statistics	Resolved	0.17.8, 0.18.4, 0.19.3	0.17.10, 0.18.5, 0.19.4

Resolved Issues in SR2

The following table lists the issues resolved in Service Release 2.

Table 9: Issues resolved in versions 0.18.5 through 0.18.5 (JIRA)

Type	Key	Summary	Resolution	Fix Version(s)
	BGPCEP-1020	PCEP accesses dead transaction chain	Done	0.18.5, 0.19.4
	BGPCEP-1021	Failure to update OpenConfig statistics	Done	0.17.10, 0.18.5, 0.19.4

Known Issues in SR2

The following table lists the known issues that exist in Service Release 2.

Resolved Issues in SR3

The following table lists the issues resolved in Service Release 3.

Table 10: Issues resolved in versions 0.18.6 through 0.18.8 (JIRA)

Type	Key	Summary	Resolution	Fix Version(s)
	BGPCEP-1024	Update restconf links in documentation	Done	0.18.7, 0.19.6, 0.20.0
	BGPCEP-1025	Rework bgpcep README to README.md	Done	0.18.7, 0.19.6, 0.20.2

Known Issues in SR3

The following table lists the known issues that exist in Service Release 3.

Controller

Overview

The Controller project is an infrastructure service that supports other OpenDaylight projects. It does not have user-facing features.

Behavior/Feature Changes

No changes.

New Features

There are no new features.

Deprecated and Removed Features

No deprecated or removed features.

Resolved Issues

The following table lists the issues resolved in this release.

Table 11: Issues resolved in versions 6.0.0 through 6.0.2 (JIRA)

Type	Key	Summary	Resolution	Fix Version(s)
	CONTROLLER-2037	Fail to serialize oversized message	Done	5.0.5, 6.0.0
	CONTROLLER-2048	Bump akka to 2.6.20	Done	4.0.13, 5.0.7, 6.0.2

Known Issues

The following table lists the known issues that exist in this release.

Table 12: Issues affecting versions 6.0.0 through 6.0.2 (JIRA)

Type	Key	Summary	Status	Affected Version(s)	Fix Version(s)
	CONTROLLER-2052	cds-access-api uses wrong ABI version	Resolved	2.0.0, 2.0.10, 3.0.0, 3.0.16, 4.0.0, 4.0.13, 5.0.0, 5.0.7, 6.0.0, 6.0.2, Magnesium, Magnesium SR3	5.0.8, 6.0.3

Resolved Issues in SR1

The following table lists the issues resolved in Service Release 1.

Table 13: Issues resolved in versions 6.0.3 through 6.0.5 (JIRA)

Type	Key	Summary	Resolution	Fix Version(s)
	CONTROLLER-2052	cds-access-api uses wrong ABI version	Done	5.0.8, 6.0.3
	CONTROLLER-2058	Define RaftVersion.ARGON	Done	5.0.8, 6.0.4, 7.0.0
	CONTROLLER-2059	Deprecate ABIVersions up to and including SODIUM_SR1	Done	6.0.4

Known Issues in SR1

The following table lists the known issues that exist in Service Release 1.

Resolved Issues in SR2

The following table lists the issues resolved in Service Release 2.

Table 14: Issues resolved in versions 6.0.6 through 6.0.7 (JIRA)

Type	Key	Summary	Resolution	Fix Version(s)
	CONTROLLER-2056	Minimize serialization proxy names in controller.datastore.persisted	Done	6.0.7, 7.0.0

Known Issues in SR2

The following table lists the known issues that exist in Service Release 2.

Resolved Issues in SR3

The following table lists the issues resolved in Service Release 3.

Table 15: Issues resolved in versions 6.0.8 through 6.0.9 (JIRA)

Type	Key	Summary	Resolution	Fix Version(s)
	CONTROLLER-2017	Tell-based protocol mis-reports standalone transaction count	Done	6.0.8, 7.0.5
	CONTROLLER-2075	Tell-based protocol should honor shard-batched-modification-count	Done	6.0.8, 7.0.5

Known Issues in SR3

The following table lists the known issues that exist in Service Release 3.

Daexim

Overview

Data Export/Import (Daexim) feature allows OpenDaylight administrators to export the current system state to the file system or to import the state from the file system.

Behavior/Feature Changes

Here is the link to the features improved in this release:

OpenDaylight JIRA Tickets - Improvement

New Features

Here is the link to the new features introduced in this release:

OpenDaylight JIRA Tickets - New Feature

Deprecated Features

Here is the link to the features removed in this release:

OpenDaylight JIRA Tickets - Deprecated Feature

Resolved Issues

Here is the link to the resolved issues fixed in this release:

OpenDaylight JIRA Tickets - Resolved Issue

Known Issues

Here is the link to the known issues exist in this release:

OpenDaylight JIRA Tickets - Known Issue

Distribution

Overview

The Distribution project is the placeholder for the ODL karaf distribution. The project currently generates 3 artifacts:

Table 16: Distribution Artifacts

Artifact	Description
Managed distribution (e.g., `karaf-<version>.tar.gz`)	This includes the managed projects in OpenDaylight (refer to, *Managed Release*).
Common distribution (e.g., `opendaylight-<version>.tar.gz`)	This includes managed and self-managed projects (refer to, *Managed Release*).
ONAP distribution (e.g., `onap-karaf-<version>.tar.gz`)	This is the distribution used in the ONAP CCSDK project.

The distribution project is also the placeholder for the distribution scripts. Example of these scripts:

- *Clustering scripts in Distribution*

Behavior/Feature Changes

Here is the link to the features improved in this release:

OpenDaylight JIRA Tickets - Improvement

New Features

Here is the link to the new features introduced in this release:

OpenDaylight JIRA Tickets - New Feature

Deprecated Features

Here is the link to the features removed in this release:

OpenDaylight JIRA Tickets - Deprecated Feature

Resolved Issues

Here is the link to the resolved issues fixed in this release:

OpenDaylight JIRA Tickets - Resolved Issue

Known Issues

Here is the link to the known issues exist in this release:

OpenDaylight JIRA Tickets - Known Issue

InfraUtils

Overview

The InfraUtils project provides a low-level utility for use by other OpenDaylight projects, including:

- @Inject DI
- `Utils` incl. `org.opendaylight.infrautils.utils.concurrent`
- Test Utilities
- Ready Service
- Integration Test Utilities (`itestutils`)
- `Diagstatus`

Behavior/Feature Changes

There are no changes to behavior.

New Features

There are no new features.

Deprecated and Removed Features

There are no deprecated or removed features.

Resolved Issues

The following table lists the issues resolved in this release.

Known Issues

The following table lists the known issues that exist in this release.

Resolved Issues in SR1

The following table lists the issues resolved in Service Release 1.

Known Issues in SR1

The following table lists the known issues that exist in Service Release 1.

Resolved Issues in SR2

The following table lists the issues resolved in Service Release 2.

Known Issues in SR2

The following table lists the known issues that exist in Service Release 2.

Table 17: Issues affecting versions 4.0.4 through 4.0.4 (JIRA)

Type	Key	Summary	Status	Affected Version(s)	Fix Version(s)
	INFRAUTILS-100	ServiceStatusSummary.toJSON() fails with upgraded Java	Resolved	3.0.5, 4.0.4, 5.0.4	4.0.7, 5.0.6, 6.0.0

Resolved Issues in SR3

The following table lists the issues resolved in Service Release 3.

Known Issues in SR3

The following table lists the known issues that exist in Service Release 3.

JSON-RPC

Overview

JSON-RPC 2.0 is a lightweight remote procedure call and notification specification maintained by JSON RPC. OpenDaylight uses the YANG-modelled JSON-RPC 2.0 specification as described in the IETF DRAFT.

Behavior/Feature Changes

Here is the link to the features improved in this release:

OpenDaylight JIRA Tickets - Improvement

New Features

Here is the link to the new features introduced in this release:

OpenDaylight JIRA Tickets - New Feature

Deprecated Features

Here is the link to the features removed in this release:

OpenDaylight JIRA Tickets - Deprecated Feature

Resolved Issues

Here is the link to the resolved issues fixed in this release:

OpenDaylight JIRA Tickets - Resolved Issue

Known Issues

Here is the link to the known issues exist in this release:

OpenDaylight JIRA Tickets - Known Issue

LISP Flow Mapping

Overview

LISP (Locator ID Separation Protocol) Flow Mapping service provides mapping services, including LISP Map-Server and LISP Map-Resolver services that store and serve mapping data to dataplane nodes and to OpenDaylight applications. Mapping data can include mapping of virtual addresses to physical network addresses where the virtual nodes are reachable or hosted. Mapping data can also include a variety of routing policies including traffic engineering and load balancing. To leverage this service, OpenDaylight applications and services can use the northbound REST API to define the mappings and policies in the LISP Mapping Service. Dataplane devices capable of LISP control protocol can leverage this service through a southbound LISP plugin. LISP-enabled devices must be configured to use this OpenDaylight service, since their Map- Server and/or Map-Resolver.

Southbound LISP plugin supports the LISP control protocol (that is, Map-Register, Map-Request, Map-Reply messages). It can also be used to register mappings in the OpenDaylight mapping service.

Behavior/Feature Changes

Here is the link to the features improved in this release:

OpenDaylight JIRA Tickets - Improvement

New Features

Here is the link to the new features introduced in this release:

OpenDaylight JIRA Tickets - New Feature

Deprecated Features

Here is the link to the features removed in this release:

OpenDaylight JIRA Tickets - Deprecated Feature

Resolved Issues

Here is the link to the resolved issues fixed in this release:

OpenDaylight JIRA Tickets - Resolved Issue

Known Issues

Here is the link to the known issues exist in this release:

OpenDaylight JIRA Tickets - Known Issue

Model-Driven Service Abstraction Layer (MD-SAL)

Overview

MD-SAL provides infrastructure for binding YANG models to Java object model and infrastructure for providing YANG-defined interaction patterns: * Reactive datastore update * RPC and Action invocation * Notification sourcing and delivery

Behavior/Feature Changes

The Binding mapping of `type identityref` YANG statement has been updated to use singleton objects instead of corresponding `.class` references. These singleton objects are exposed as `VALUE` constants, hence the migration is a straightforward move from `Foo.class` to `Foo.VALUE`.

Binding classes generated for `type enumeration` YANG statement now have `ofName()` and `ofValue()` methods, which return a non-null object or throw an `IllegalArgumentException`.

The `feature` YANG statement now has a representation in Binding: it is a final class which subclasses `YangFeature` with a singleton value.

New Features

Both `DOMNotificationService` and `NotificationService` have gained the ability to register single-type and flexible multi-type notification listerners. This change allows Binding users to implement only specific listeners, unlike the previous method of using generated interfaces extending `NotificationListener` – which required implementation of methods for every notification defined in a specific module.

Deprecated Features

`NotificationService.registerNotificationListener()` method has been deprecated, along with its Binding specification interfaces based on `NotificationListener`. These will be removed in the next major release.

Removed Features

No removed features.

Resolved Issues

The following table lists the issues resolved in this release.

Table 18: Issues resolved in versions 10.0.0 through 10.0.2 (JIRA)

Type	Key	Summary	Resolution	Fix Version(s)
	MDSAL-756	PingPongTransactionChain's cancel() always closes underlying chain	Done	10.0.0, 8.0.14, 9.0.3
	MDSAL-747	Do not shade byte-buddy	Done	10.0.0
	MDSAL-733	Change 'type identityref' Binding representation to normal objects	Done	10.0.0
	MDSAL-641	Convert mdsal-binding-dom-codec to a JPMS module	Done	10.0.0
	MDSAL-754	Generate ofName() and ofValue() for enumerations	Done	10.0.0
	MDSAL-753	Generate a switch expression for enum's forName()/forValue() methods	Done	10.0.0
	MDSAL-740	Generate fields for all Identityrefs in an Union binding class.	Done	10.0.0
	MDSAL-692	Use HexFormat to print out byte[] properties	Done	10.0.0
	MDSAL-693	Improve TypeObject.toString()	Done	10.0.0
	MDSAL-755	Improve javadoc of generated classes	Done	10.0.0
	MDSAL-760	Generate javadoc for unions	Done	10.0.1
	MDSAL-761	Generate javadoc for augments	Done	10.0.1
	MDSAL-759	Improve javadoc for generated keys	Done	10.0.1
	MDSAL-758	Generate javadocs for Builder.withKey()	Done	10.0.1
	MDSAL-762	Generate javadoc for ScalarTypes	Done	10.0.2
	MDSAL-769	Reflect identity status in generated classes	Done	10.0.2, 9.0.6
	MDSAL-770	Reflect feature status in generated classes	Done	10.0.2
	MDSAL-701	Support for atomic registration of diverse DOMNotificationListeners	Done	10.0.0
	MDSAL-702	Add support for listening on multiple notifications	Done	10.0.0
	MDSAL-767	Add support for advertizing supported features	Done	10.0.1
	MDSAL-766	Generate code for 'feature' statements	Done	10.0.1
	MDSAL-49	Do not generate Builders for Union types	Done	10.0.0
	MDSAL-757	Do not generate @java.beans.ConstructorProperties	Done	10.0.0
	MDSAL-704	Do not use IllegalArgumentCodec	Done	10.0.0
	MDSAL-496	Deprecate generated notification listener interface	Done	10.0.0

Known Issues

The following table lists the known issues that exist in this release.

Table 19: Issues affecting versions 10.0.0 through 10.0.2 (JIRA)

Type	Key	Summary	Status	Affected Version(s)	Fix Version(s)
	MDSAL-796	DOMRpcService.registerImplementations() does not work with pre-existing entries	Resolved	10.0.0, 10.0.3, 11.0.0, 7.0.0, 7.0.14, 8.0.0, 8.0.16, 9.0.0, 9.0.6	10.0.4, 11.0.1, 9.0.7
	MDSAL-828	Fix mdsal-binding-dom-codec module definition	Resolved	10.0.0, 10.0.8, 11.0.0, 11.0.11	11.0.12, 12.0.0

Resolved Issues in SR1

The following table lists the issues resolved in Service Release 1.

Table 20: Issues resolved in versions 10.0.3 through 10.0.5 (JIRA)

Type	Key	Summary	Resolution	Fix Version(s)
	MDSAL-796	DOMRpcService.registerImplementations() does not work with pre-existing entries	Done	10.0.4, 11.0.1, 9.0.7
	MDSAL-791	Do not track uninteresting bundles	Done	10.0.3, 11.0.0
	MDSAL-792	Union value classes need to enforce non-null components	Done	10.0.4, 11.0.0

Known Issues in SR1

The following table lists the known issues that exist in Service Release 1.

Resolved Issues in SR2

The following table lists the issues resolved in Service Release 2.

Known Issues in SR2

The following table lists the known issues that exist in Service Release 2.

Table 21: Issues affecting versions 10.0.6 through 10.0.6 (JIRA)

Type	Key	Summary	Status	Affected Version(s)	Fix Version(s)
	MDSAL-811	Notification registration not updated with 'registerNotificationListeners'	Resolved	10.0.6, 11.0.5	10.0.7, 11.0.6
	MDSAL-818	Wildcard KeyedInstanceIdentifier cannot be constructed	Resolved	10.0.6, 3.0.16, 4.0.17, 5.0.17, 6.0.12, 7.0.14, 9.0.8	10.0.7, 11.0.8, 12.0.0
	MDSAL-822	DOMDataTreeCommitCohort does not expose EffectiveModelContext	Resolved	10.0.6, 11.0.7	11.0.8, 12.0.0

Resolved Issues in SR3

The following table lists the issues resolved in Service Release 3.

Table 22: Issues resolved in versions 10.0.7 through 10.0.8 (JIRA)

Type	Key	Summary	Resolution	Fix Version(s)
	MDSAL-811	Notification registration not updated with 'registerNotificationListeners'	Done	10.0.7, 11.0.6
	MDSAL-818	Wildcard KeyedInstanceIdentifier cannot be constructed	Done	10.0.7, 11.0.8, 12.0.0
	MDSAL-824	Runtime types fail to be generated for ietf-keystore	Done	10.0.8, 11.0.10, 12.0.0
	MDSAL-806	Add Augmentable.augmentationOrElseThrow()	Done	10.0.7, 11.0.5
	MDSAL-821	Annotate generated serialVersionUID with @java.io.Serial	Done	10.0.7, 11.0.8, 12.0.0

Known Issues in SR3

The following table lists the known issues that exist in Service Release 3.

Table 23: Issues affecting versions 10.0.7 through 10.0.8 (JIRA)

Type	Key	Summary	Status	Affected Version(s)	Fix Version(s)
	MDSAL-828	Fix mdsal-binding-dom-codec module definition	Resolved	10.0.0, 10.0.8, 11.0.0, 11.0.11	11.0.12, 12.0.0
	MDSAL-838	AbstractPingPongTransactionChain is not idempotent	Resolved	10.0.8, 11.0.12, 12.0.1, 3.0.16, 4.0.17, 5.0.17, 6.0.12, 7.0.14, 8.0.16, 9.0.8	10.0.9, 11.0.13, 12.0.2

NETCONF

Overview

The NETCONF projects hosts multiple components relating to IETF's NETCONF Working Group:

- Northbound and southbound plugins for NETCONF protocol, as described in RFC-6241
- Northbound plugin for RESTCONF protocol, as described in RFC-8040
- Northnound plugin for descring RESTCONF endpoint in terms of OpenAPI 3.0

Behavior/Feature Changes

The `odl-restconf-nb-rfc8040` feature has been renamed to `odl-restconf-nb`.

New Features

There are no new features.

Deprecated and Removed Features

The old RESTCONF endpoint `localhost:8181/restconf`, as installed via `odl-restconf-nb-bierman02` feature, has been removed.

Resolved Issues

The following table lists the issues resolved in this release.

Table 24: Issues resolved in versions 4.0.0 through 4.0.2 (JIRA)

Type	Key	Summary	Resolution	Fix Version(s)
	NETCONF-872	Missing stack enter in sal-rest-docgen Actions	Done	3.0.2, 4.0.0
	NETCONF-877	Fail to process PATCH to data root with a top-level container as target	Done	3.0.3, 4.0.0
	NETCONF-875	Netconf device mount with hostname in payload	Done	2.0.16, 3.0.4, 4.0.0
	NETCONF-886	Typo in content parameter error-message	Done	3.0.6, 4.0.0
	NETCONF-879	netconf-mdsal fails and affects both testtool and ODL as netconf server	Done	3.0.5, 4.0.0
	NETCONF-881	Device with augmented NETCONF monitoring response cannot be connected without models inside cache/schema	Done	3.0.6, 4.0.0
	NETCONF-899	Northbound NETCONF server connection error	Done	3.0.6, 4.0.2
	NETCONF-884	Maven shade plugin replace netconf-testtol with scale-util inside release artifacts	Done	3.0.6, 4.0.0
	NETCONF-681	Netconf Callhome SSH: drop connections	Done	3.0.6, 4.0.2
	NETCONF-837	Remove restconf-nb-bierman02	Done	4.0.0
	NETCONF-897	Allow netconf-client's maximum chunk to be controlled	Done	4.0.1
	NETCONF-660	RFC 8040 query fails to return field if subfields specified for another element	Done	3.0.6, 4.0.1
	NETCONF-888	Make netconf maximum chunk size value configurable	Done	3.0.6, 4.0.1
	NETCONF-898	Allow netconf-impl's maximum chunk to be controlled	Done	4.0.2
	NETCONF-889	Improve EOM aggregator performance	Done	2.0.17, 3.0.6, 4.0.0
	NETCONF-814	Package RFC8639 ietf-subscribed-notifications module	Done	4.0.0
	NETCONF-890	Reorganize IETF model placement and naming	Done	4.0.0
	NETCONF-891	Rework ietf-netconf-monitoring-extension	Done	4.0.0
	NETCONF-893	Rename restconf-nb-rfc8040	Done	4.0.0

Known Issues

The following table lists the known issues that exist in this release.

Table 25: Issues affecting versions 4.0.0 through 4.0.2 (JIRA)

Type	Key	Summary	Status	Affected Version(s)	Fix Version(s)
	NETCONF-883	Fail to generate API Doc	Resolved	3.0.0, 3.0.4, 4.0.0, 4.0.2	3.0.7, 4.0.3
	NETCONF-905	NetconfSessionPromise reconnection failure	Resolved	3.0.6, 4.0.2	2.0.17, 3.0.7, 4.0.3
	NETCONF-827	Mount loop when setting too low connection-timeout	Resolved	2.0.14, 2.0.16, 3.0.0, 3.0.6, 4.0.0, 4.0.2	2.0.17, 3.0.7, 4.0.3
	NETCONF-887	Netconf callhome failed for devices with old KEX algorithms (SHA1)	Resolved	2.0.11, 2.0.17, 3.0.0, 3.0.8, 4.0.0, 4.0.5, 5.0.0, 5.0.1	3.0.9, 4.0.6, 5.0.2
	NETCONF-972	yanglib does not support RFC6020 media types	Resolved	3.0.0, 3.0.8, 4.0.0, 4.0.5, 5.0.0, 5.0.3	3.0.9, 4.0.6, 5.0.4

Resolved Issues in SR1

The following table lists the issues resolved in Service Release 1.

Table 26: Issues resolved in versions 4.0.3 through 4.0.4 (JIRA)

Type	Key	Summary	Resolution	Fix Version(s)
	NETCONF-883	Fail to generate API Doc	Done	3.0.7, 4.0.3
	NETCONF-905	NetconfSessionPromise reconnection failure	Done	2.0.17, 3.0.7, 4.0.3
	NETCONF-827	Mount loop when setting too low connection-timeout	Done	2.0.17, 3.0.7, 4.0.3
	NETCONF-900	mdsal-netconf-ssh fails to shutdown	Done	3.0.7, 4.0.3
	NETCONF-820	Fail to process RESTCONF fields on the mounted device	Done	4.0.3
	NETCONF-896	Improve logging inside AsyncSshHandler	Done	4.0.3
	NETCONF-894	Update sshd to 2.9.1	Done	3.0.7, 4.0.3

Known Issues in SR1

The following table lists the known issues that exist in Service Release 1.

Table 27: Issues affecting versions 4.0.3 through 4.0.4 (JIRA)

Type	Key	Summary	Status	Affected Version(s)	Fix Version(s)
	NETCONF-976	The required fields are created, but they are not populate in the Swagger API	Resolved	4.0.3, 5.0.0	4.0.9, 5.0.7, 6.0.0
	NETCONF-926	Duplicate yang-ext:mount in error message	Resolved	3.0.8, 4.0.3	3.0.9, 4.0.5, 5.0.0
	NETCONF-934	Fail to generate <discard-changes> RPC	Resolved	3.0.8, 4.0.4	3.0.9, 4.0.5, 5.0.0
	NETCONF-933	'commit' sent to devices without 'candidate' capability	Resolved	3.0.8, 4.0.4	3.0.9, 4.0.5, 5.0.0
	NETCONF-909	Fix odl-yanglib feature usage	Resolved	4.0.3	3.0.9, 4.0.6, 5.0.2
	NETCONF-910	Odl-yanglib fails to register provided sources	Resolved	4.0.3	6.0.0

Resolved Issues in SR2

The following table lists the issues resolved in Service Release 2.

Table 28: Issues resolved in versions 4.0.5 through 4.0.5 (JIRA)

Type	Key	Summary	Resolution	Fix Version(s)
	NETCONF-926	Duplicate yang-ext:mount in error message	Done	3.0.9, 4.0.5, 5.0.0
	NETCONF-934	Fail to generate <discard-changes> RPC	Done	3.0.9, 4.0.5, 5.0.0
	NETCONF-933	'commit' sent to devices without 'candidate' capability	Done	3.0.9, 4.0.5, 5.0.0
	NETCONF-927	Simplify OperationsContext with yangtools 7.0.9 +	Done	3.0.9, 4.0.5, 5.0.0

Known Issues in SR2

The following table lists the known issues that exist in Service Release 2.

Table 29: Issues affecting versions 4.0.5 through 4.0.5 (JIRA)

Type	Key	Summary	Status	Affected Version(s)	Fix Version(s)
	NETCONF-887	Netconf callhome failed for devices with old KEX algorithms (SHA1)	Resolved	2.0.11, 2.0.17, 3.0.0, 3.0.8, 4.0.0, 4.0.5, 5.0.0, 5.0.1	3.0.9, 4.0.6, 5.0.2
	NETCONF-985	Incorrect processing of RESTCONF fields for duplicate node names on NETCONF device	Resolved	3.0.9, 4.0.5, 5.0.4	4.0.8, 5.0.7, 6.0.0
	NETCONF-972	yanglib does not support RFC6020 media types	Resolved	3.0.0, 3.0.8, 4.0.0, 4.0.5, 5.0.0, 5.0.3	3.0.9, 4.0.6, 5.0.4
	NETCONF-747	Unable execute YANG patch request when targeting augmented element	Resolved	1.13.0, 3.0.8, 4.0.5, 5.0.2, Aluminium SR1, Magnesium SR2	3.0.9, 4.0.6, 5.0.3
	NETCONF-1051	SSE with sub identifier does not work	In Review	4.0.5, 5.0.4	4.0.9, 5.0.8, 6.0.3, 7.0.0

Resolved Issues in SR3

The following table lists the issues resolved in Service Release 3.

Table 30: Issues resolved in versions 4.0.6 through 4.0.8 (JIRA)

Type	Key	Summary	Resolution	Fix Version(s)
	NETCONF-887	Netconf callhome failed for devices with old KEX algorithms (SHA1)	Done	3.0.9, 4.0.6, 5.0.2
	NETCONF-953	Unable to load org.eclipse.jetty.websocket.server.WebSocketServer	Done	4.0.6, 5.0.3
	NETCONF-970	Websocket timeout exception	Done	4.0.6, 5.0.3
	NETCONF-977	OpenAPI: Schemas are not shown	Done	3.0.9, 4.0.6, 5.0.4
	NETCONF-1001	Notification eventInstant() not propagated to NotificationMessage	Done	4.0.7, 5.0.7, 6.0.0
	NETCONF-1018	NetconfNotification confuses its namespace	Done	4.0.7, 5.0.7, 6.0.0
	NETCONF-929	Cannot retrieve operation resource	Done	4.0.7, 5.0.7, 6.0.0
	NETCONF-928	Discover mountpoint RPC operation through RESTCONF is failing with response 500	Done	4.0.7, 5.0.7, 6.0.0
	NETCONF-996	OpenAPI: Fix invalid swagger schema	Done	4.0.7, 5.0.7, 6.0.0
	NETCONF-1022	OpenApi: Missing required parameters in swagger schema	Done	4.0.7, 5.0.7, 6.0.0
	NETCONF-985	Incorrect processing of RESTCONF fields for duplicate node names on NETCONF device	Done	4.0.8, 5.0.7, 6.0.0
	NETCONF-972	yanglib does not support RFC6020 media types	Done	3.0.9, 4.0.6, 5.0.4
	NETCONF-909	Fix odl-yanglib feature usage	Done	3.0.9, 4.0.6, 5.0.2
	NETCONF-747	Unable execute YANG patch request when targeting augmented element	Done	3.0.9, 4.0.6, 5.0.3
	NETCONF-28	Netconf candidate capability non RFC compliant fallback	Won't Do	4.0.6, 5.0.1
	NETCONF-988	Convert xml pretty print	Done	4.0.6, 5.0.5
	NETCONF-946	Update NETCONF user guide	Done	4.0.6, 5.0.5
	NETCONF-1038	Align netconf's documentation version with distro	Done	4.0.7, 5.0.7, 6.0.0

Known Issues in SR3

The following table lists the known issues that exist in Service Release 3.

Table 31: Issues affecting versions 4.0.6 through 4.0.8 (JIRA)

Type	Key	Summary	Status	Affected Version(s)	Fix Version(s)
	NETCONF-1095	Fail to process PATCH with target containing a single forward slash	Resolved	4.0.8, 5.0.6, 6.0.0	5.0.8, 6.0.2, 7.0.0
	NETCONF-1152	RESTCONF DataTreeChange notifications use incorrect format	Resolved	2.0.17, 3.0.9, 4.0.8, 5.0.7, 6.0.2	7.0.0
	NETCONF-1130	POST returns 500 on data already exists	In Progress	4.0.8, 5.0.7, 6.0.1	7.0.0
	NETCONF-1170	netconf-client-mdsal emits useless namespace in filter	In Review	3.0.9, 4.0.8, 5.0.7, 6.0.4	4.0.9, 5.0.8, 6.0.5, 7.0.0

OpenFlow Plugin

Overview

The OpenFlow Plugin project provides the following functionality:

- **OpenFlow 1.0/1.3 implementation project**: This provides the implementation of the OpenFlow 1.0 and OpenFlow 1.3 specification.
- **ONF Approved Extensions Project**: This provides the implementation of following ONF OpenFlow 1.4 feature, which is approved as extensions for the OpenFlow 1.3 specification:
  - **Nicira Extensions Project**: This provides the implementation of the `Nicira` extensions. Some of the important extensions implemented are **Connection Tracking Extension** and **Group Add-Mod Extension**
- **OpenFlow-Based Applications Project**: This provides the following applications that user can leverage out-of-the-box in developing their application or as a direct end consumer:

Table 32: OpenFlow-Based Applications

Application	Description
Forwarding Rules Manager	Application provides functionality to add/remove/update flow/groups/meters.
LLDP Speaker	Application sends periodic LLDP packet out on each OpenFlow switch port for link discovery.
Topology LLDP Discovery	Application intercept the LLDP packets and discover the link information.
Topology Manager	Application receives the discovered links information from Topology LLDP Discovery application and stores in the topology yang model datastore.
Reconciliation Framework	Framework that exposes the APIs that consumer application (in-controller) can leverage to participate in the switch reconciliation process in the event of switch connection/reconnection.
Arbitrator Reconciliation	Application exposes the APIs that consumer application or direct user can leverage to trigger the device configuration reconciliation.
OpenFlow Java Library Project	Provides the OpenFlow Java Library that converts the data based on OpenFlow plugin data models to the OpenFlow java models before sending it down the wire to the device.
Reconciliation	Reconciles the state using OpenFlow 1.4 bundles.

Behavior/Feature Changes

Here is the link to the features improved in this release:

OpenDaylight JIRA Tickets - Improvement

New Features

Here is the link to the new features introduced in this release:

OpenDaylight JIRA Tickets - New Feature

Deprecated Features

Here is the link to the features removed in this release:

OpenDaylight JIRA Tickets - Deprecated Feature

Resolved Issues

Here is the link to the resolved issues fixed in this release:

OpenDaylight JIRA Tickets - Resolved Issue

Known Issues

Here is the link to the known issues exist in this release:

OpenDaylight JIRA Tickets - Known Issue

OVSDB

Overview

The OVSDB Project provides the following functionality:

- The OVSDB southbound plugin handles any OVS device that supports both the OVSDB schema and uses the OVSDB protocol. This feature implements a defined YANG model. Developers can use this functionality to develop in-controller applications that leverage OVSDB for device configuration.
- The HWvTep southbound plugin handles any OVS device that supports both the OVSDB hardware vTEP schema and uses OVSDB protocol. This feature implements a defined YANG model. Developers can use this functionality to develop in-controller applications that leverage OVSDB hardware vTEP plugin for device configuration.

Behavior/Feature Changes

Here is the link to the features improved in this release:

OpenDaylight JIRA Tickets - Improvement

New Features

Here is the link to the new features introduced in this release:

OpenDaylight JIRA Tickets - New Feature

Deprecated Features

Here is the link to the features removed in this release:

OpenDaylight JIRA Tickets - Deprecated Feature

Resolved Issues

Here is the link to the resolved issues fixed in this release:

OpenDaylight JIRA Tickets - Resolved Issue

Known Issues

Here is the link to the known issues exist in this release:

OpenDaylight JIRA Tickets - Known Issue

ServiceUtils

Overview

The ServiceUtils infrastructure project is a utility to assist in the operation and maintenance of services provided by OpenDaylight. A service is a functionality provided by the ODL controller as seen by the operator. Services can be either networking services (e.g. L2, L3/VPN, NAT, etc.) or infra services (e.g. OpenFlow). Services are also provided by the different ODL projects, such as Netvirt, Genius and OpenFlow plugin, and are comprised of a set of Java Karaf bundles and associated MD-SAL datastores.

Behavior/Feature Changes

Here is the link to the features improved in this release:

OpenDaylight JIRA Tickets - Improvement

New Features

Here is the link to the new features introduced in this release:

OpenDaylight JIRA Tickets - New Feature

Deprecated Features

Here is the link to the features removed in this release:

OpenDaylight JIRA Tickets - Deprecated Feature

Resolved Issues

Here is the link to the resolved issues fixed in this release:

OpenDaylight JIRA Tickets - Resolved Issue

Known Issues

Here is the link to the known issues exist in this release:

OpenDaylight JIRA Tickets - Known Issue

Transport PCE

Overview

Transport PCE is an application running on top of the OpenDaylight controller. Its primary function is to control an optical transport infrastructure using a non-proprietary South Bound Interface (SBI).

The controlled transport infrastructure includes a WDM (Wave Division Multiplexing) layer and an OTN (optical transport network) layer. The WDM layer is built from ROADMs (reconfigurable optical add-drop multiplexer) with colorless, directionless and contention-less features. The OTN layer is built from transponders, muxponders or switchponders which include OTN switching functionalities.

Transport PCE leverages OpenROADM Multi-Source-Agreement (MSA), which defines interoperability specifications, consisting of both optical interoperability and YANG data models.

The TransportPCE implementation includes:

Table 33: Transport PCE implementation

Feature	Description
Northbound API	These APIs are for higher level applications, implemented in the Service Handler bundle. It relies on the service model defined in the MSA. A minimal experimental support of TAPI topology is also proposed but is not installed by default.
Renderer and OLM	The renderer and OLM (Optical Line Management) bundles allow configuring OpenROADM devices through a southbound NETCONF/YANG interface (based on the MSA device models). This release supports the OpenROADM devices version 1.2.1 version 2.2.1.
Topology Management	This feature is based on the defined MSA network model.
Path Calculation Engine (PCE)	PCE here has a different meaning than the BGPCEP project since it is not based on (G)MPLS. This bundle allows to compute path across the topology to create services. Impairment aware path computation can be delegated to a GNPy server (hardcoded server address configuration and limited support at that time)
Inventory	This feature is not installed by default. It proposes an experimental support for an external inventory DB currently limited to 1.2.1 OpenROADM devices.

The internal RPCs between those modules are defined in the Transport Service Path models.

Behavior/Feature Changes

TBD

Changes planned in Sulfur release stream

TBD

New Features

TBD

Deprecated Features

There are no deprecated or removed features.

Resolved Issues

The following table lists the issues resolved in this release.

Table 34: Issues resolved in versions Sulfur through Sulfur (JIRA)

Type	Key	Summary	Resolution	Fix Version(s)
	TRNSPRTPCE-645	Bug in the ethernet loop of the 221 E2E tests	Done	Phosphorus, Sulfur
	TRNSPRTPCE-677	Fix PCE pruning phase	Done	Sulfur
	TRNSPRTPCE-675	create-service check	Done	Sulfur
	TRNSPRTPCE-610	2.2.1 port-capabilities support in port-mapping	Done	Phosphorus, Sulfur
	TRNSPRTPCE-596	Port-mapping fails for unsupported if-cap types	Done	Phosphorus, Sulfur
	TRNSPRTPCE-598	Incorrect portmapping for SRGs with multiple circuit packs	Done	Phosphorus, Sulfur
	TRNSPRTPCE-589	Change the device renderer result response for 400GE service	Duplicate	Sulfur
	TRNSPRTPCE-597	Refactor networkmodel util OpenRoadmOtnTopology	Done	Phosphorus, Sulfur
	TRNSPRTPCE-647	Update XPDR-C1 configuration file with a list of xponder	Done	Phosphorus, Sulfur
	TRNSPRTPCE-646	Update checks for service requests in service handler	Done	Phosphorus, Sulfur
	TRNSPRTPCE-643	Manage 100GE service terminated on a 2.2.1 XPDR with a list of xponder	Done	Phosphorus, Sulfur
	TRNSPRTPCE-572	Refactor PceOtnNode class	Done	Phosphorus, Sulfur
	TRNSPRTPCE-516	Network model 10.1	Done	Sulfur
	TRNSPRTPCE-515	Service Model 10.1	Done	Sulfur
	TRNSPRTPCE-376	PCE-GNPy adaptation for higher rates	Done	Phosphorus, Sulfur
	TRNSPRTPCE-513	Update GNPy REST implementation	Done	Phosphorus, Sulfur
	TRNSPRTPCE-639	Prepare Migration to Sulfur	Done	Sulfur
	TRNSPRTPCE-679	Replace transportpce-topology yang by existing OpenROADM models	Done	Sulfur
	TRNSPRTPCE-676	Implement autonomous service rerouting	Done	Sulfur
	TRNSPRTPCE-606	Change the output response of B100G interface creation result	Done	Phosphorus, Sulfur
	TRNSPRTPCE-583	Complement functional tests to include notifications	Done	Sulfur
	TRNSPRTPCE-567	Functional tests RFC8040 REST API migration	Done	Phosphorus, Sulfur

Known Issues

The following table lists the known issues that exist in this release.

Table 35: Issues affecting versions Sulfur through Sulfur (JIRA)

Type	Key	Summary	Status	Affected Version(s)	Fix Version(s)
	TRNSPRTP(701	karaf client fails in karaf-4.3.8 and 4.4.2	Verified	Argon, Chlorine, Sulfur	Potassium
	TRNSPRTP(595	Wrong HTTP code returned by some device+OTN renderer func tests 7.1 with RFC8040 URLs API	Verified	Phosphorus, Sulfur	Phosphorus

YANG Tools

Overview

YANG Tools provides a set of libraries to deal with YANG models and data modeled using them.

Behavior/Feature Changes

`ImmutableOffsetMap` and `SingletonSet` classes now use the Serialization Proxy Pattern, improving their serialization footprint.

A large number of abstract classes and interfaces not intended to be directly subclasses/implemented are now `sealed`. This results in better API definitions and provides a clearer guidance to users. This change also means these can no longer be mocked through Mockito and similar frameworks. Users are advised to use concrete implementations instead.

New Features

YANG Parser now supports the `module-tag` as defined in RFC819.

Removed Features

The `CheckedBuilder` and `Builder` concepts have been removed in this release.

The `IllegalArgumentCodec` concept has been removed in this release.

The support for resolving inter-module dependencies based on semantic version has been removed.

Deprecated Features

Code generation plugin APIs for `yang-maven-plugin` contained in the `yang-maven-plugin-spi` artifact are deprecated and will be removed in the next major release. Their replacement live in maven-agnostic `plugin-generator-api`.

The `SchemaPath` class has been deprecated and will be removed in the next major release. Please use its correct replacements, `SchemaNodeIdentifier` and `EffectiveStatementInference`, which provide more powerful capabilities.

Resolved Issues

The following table lists the issues resolved in this release.

Table 36: Issues resolved in versions 9.0.0 through 9.0.1 (JIRA)

Type	Key	Summary	Resolution	Fix Version(s)
	YANGTOOLS-1428	Missing failedSource in SchemaResolutionException	Done	7.0.16, 8.0.4, 9.0.0
	YANGTOOLS-1431	Unsupported leaf under causes parsing failure	Done	8.0.4, 9.0.0
	YANGTOOLS-1433	YangInstanceIdentifierWriter does not handle nested augmentations	Done	8.0.5, 9.0.0
	YANGTOOLS-1438	Decimal64.toString() loses negative sign	Done	8.0.6, 9.0.0
	YANGTOOLS-1434	Building SchemaContext fails when augmenting submodel data	Done	8.0.6, 9.0.0
	YANGTOOLS-1436	Unexpected error while processing submodule references	Done	10.0.0, 8.0.7, 9.0.1
	YANGTOOLS-1322	Remove concepts.Builder and concepts.CheckedBuilder	Done	9.0.0
	YANGTOOLS-1332	Remove concepts.IllegalArgumentCodec	Done	9.0.0
	YANGTOOLS-1425	Compute YangInstanceIdentifier.hashCode lazily	Done	9.0.0
	YANGTOOLS-1420	Update known Unicode blocks for Java 17	Done	9.0.0
	YANGTOOLS-1424	Switch yang-model-validator to argparse4j	Done	9.0.0
	YANGTOOLS-1440	Add support for adjusting Decimal64 scale	Done	8.0.6, 9.0.0
	YANGTOOLS-1437	Normalize Decimal64 scale in DecimalStringCodec	Done	8.0.6, 9.0.0
	YANGTOOLS-1422	Invert SchemaInferenceStack's deque	Done	9.0.0
	YANGTOOLS-837	Add support for varied model conformance	Done	9.0.0
	YANGTOOLS-1396	Refactor IfFeaturePredicateVisitor	Done	9.0.0
	YANGTOOLS-1439	Improve Decimal64.toString() implementation	Done	9.0.1
	YANGTOOLS-1449	Convert yang-data-codec-gson into a JPMS	Done	9.0.1
	YANGTOOLS-1315	Add support for RFC8819 "module-tag" extension	Done	9.0.1
	YANGTOOLS-1451	Deprecate XMLStreamNormalizedNodeStreamWriter.create() with SchemaPath	Done	9.0.1
	YANGTOOLS-1450	Deprecate JSONNormalizedNodeStreamWriter.create*Writer() with SchemaPath	Done	9.0.1
	YANGTOOLS-1432	Remove support for semantic version imports	Done	9.0.0
	YANGTOOLS-1427	Do not fallback to toString() in UnionStringCodec	Done	9.0.0

Known Issues

The following table lists the known issues that exist in this release.

Table 37: Issues affecting versions 9.0.0 through 9.0.1 (JIRA)

Type	Key	Summary	Status	Affected Version(s)	Fix Version(s)
	YANGTOOL-1443	Fix YangDataEffectiveStatement definition	Resolved	7.0.17, 8.0.6, 9.0.0	10.0.0, 8.0.8, 9.0.2
	YANGTOOL-1455	Data change event notification fails	Resolved	7.0.0, 7.0.17, 8.0.0, 8.0.7, 9.0.0, 9.0.2	10.0.0, 7.0.18, 8.0.8, 9.0.2
	YANGTOOL-1467	yang-xpath-impl depends on javax.inject	Resolved	10.0.0, 8.0.0, 8.0.8, 9.0.0, 9.0.2	10.0.1, 8.0.9, 9.0.3
	YANGTOOL-1465	Unexpected error processing source SourceIdentifier [ietf-network@2018-02-26]	Resolved	10.0.0, 8.0.0, 8.0.8, 9.0.0, 9.0.2	10.0.1, 8.0.9, 9.0.4
	YANGTOOL-1473	XML/JSON YangInstanceIdentifier codecs mis-handle key values	Resolved	10.0.4, 7.0.18, 8.0.0, 9.0.0	10.0.5, 11.0.0, 8.0.10, 9.0.7
	YANGTOOL-1514	Failed to process YANGs containing refine of a if-feature'd target	Resolved	10.0.0, 10.0.7, 7.0.18, 8.0.10, 9.0.0, 9.0.8	10.0.8, 11.0.0, 9.0.9

Resolved Issues in SR1

The following table lists the issues resolved in Service Release 1.

Table 38: Issues resolved in versions 9.0.2 through 9.0.5 (JIRA)

Type	Key	Summary	Resolution	Fix Version(s)
	YANGTOOLS-1443	Fix YangDataEffectiveStatement definition	Done	10.0.0, 8.0.8, 9.0.2
	YANGTOOLS-1455	Data change event notification fails	Done	10.0.0, 7.0.18, 8.0.8, 9.0.2
	YANGTOOLS-1448	Failed to parse deviation statement present in submodule	Done	10.0.1, 8.0.9, 9.0.3
	YANGTOOLS-1445	Node name collision for unique argument	Done	10.0.1, 8.0.9, 9.0.3
	YANGTOOLS-1467	yang-xpath-impl depends on javax.inject	Done	10.0.1, 8.0.9, 9.0.3
	YANGTOOLS-1465	Unexpected error processing source SourceIdentifier [ietf-network@2018-02-26]	Done	10.0.1, 8.0.9, 9.0.4
	YANGTOOLS-1469	NormalizedNodeStreamWriter.create() ignores path	Done	10.0.2, 9.0.5
	YANGTOOLS-1487	DOMSourceAnydata marked as private and hence unusable	Won't Do	9.0.5

Known Issues in SR1

The following table lists the known issues that exist in Service Release 1.

Table 39: Issues affecting versions 9.0.2 through 9.0.5 (JIRA)

Type	Key	Summary	Status	Affected Version(s)	Fix Version(s)
	YANGTOOLS-1470	Fail to find unique argument node in augmented list	Resolved	10.0.1, 8.0.9, 9.0.5	10.0.3, 8.0.10, 9.0.6
	YANGTOOLS-1471	Fail to process unsupported augment statement	Resolved	10.0.1, 8.0.9, 9.0.5	10.0.3, 8.0.10, 9.0.6

Resolved Issues in SR2

The following table lists the issues resolved in Service Release 2.

Table 40: Issues resolved in versions 9.0.6 through 9.0.6 (JIRA)

Type	Key	Summary	Resolution	Fix Version(s)
	YANGTOOLS-1470	Fail to find unique argument node in augmented list	Done	10.0.3, 8.0.10, 9.0.6
	YANGTOOLS-1471	Fail to process unsupported augment statement	Done	10.0.3, 8.0.10, 9.0.6

Known Issues in SR2

The following table lists the known issues that exist in Service Release 2.

Table 41: Issues affecting versions 9.0.6 through 9.0.6 (JIRA)

Type	Key	Summary	Status	Affected Version(s)	Fix Version(s)
	YANGTOOLS-1474	Fail to process augmentation with empty grouping	Resolved	10.0.2, 8.0.9, 9.0.6	10.0.3, 8.0.10, 9.0.7
	YANGTOOLS-1481	Fail to parse a list with unique statement when unsupported	Resolved	10.0.3, 8.0.9, 9.0.6	10.0.4, 8.0.10, 9.0.7
	YANGTOOLS-1480	Fail to process deviation/augmentation from multiple modules	Resolved	10.0.3, 7.0.18, 8.0.9, 9.0.6	10.0.5, 11.0.0, 8.0.10, 9.0.7
	YANGTOOLS-1485	Fail to process deviation of augmented node without feature support	Resolved	10.0.3, 8.0.9, 9.0.6	10.0.5, 11.0.0, 8.0.10, 9.0.7

Resolved Issues in SR3

The following table lists the issues resolved in Service Release 3.

Table 42: Issues resolved in versions 9.0.7 through 9.0.8 (JIRA)

Type	Key	Summary	Resolution	Fix Version(s)
	YANGTOOLS-1458	Instance identifier parser : handle inner quotes within quoted strings	Done	10.0.3, 8.0.10, 9.0.7
	YANGTOOLS-1361	YangXPathExpression.interpretAsInstanceIdentifier() does not handle unqualified names	Done	10.0.3, 8.0.10, 9.0.7
	YANGTOOLS-1474	Fail to process augmentation with empty grouping	Done	10.0.3, 8.0.10, 9.0.7
	YANGTOOLS-1481	Fail to parse a list with unique statement when unsupported	Done	10.0.4, 8.0.10, 9.0.7
	YANGTOOLS-1011	Empty XML anydata nodes cannot be normalized	Done	10.0.5, 11.0.0, 8.0.10, 9.0.7
	YANGTOOLS-1480	Fail to process deviation/augmentation from multiple modules	Done	10.0.5, 11.0.0, 8.0.10, 9.0.7
	YANGTOOLS-1473	XML/JSON YangInstanceIdentifier codecs mis-handle key values	Done	10.0.5, 11.0.0, 8.0.10, 9.0.7
	YANGTOOLS-1485	Fail to process deviation of augmented node without feature support	Done	10.0.5, 11.0.0, 8.0.10, 9.0.7
	YANGTOOLS-1505	yang-maven-plugin elicits deprecation warning with maven-3.9.1	Done	10.0.7, 11.0.0, 9.0.8
	YANGTOOLS-1486	Reduce AbstractPrerequisite footprint	Done	10.0.4, 8.0.10, 9.0.7

Known Issues in SR3

The following table lists the known issues that exist in Service Release 3.

Table 43: Issues affecting versions 9.0.7 through 9.0.8 (JIRA)

Type	Key	Summary	Status	Affected Version(s)	Fix Version(s)
	YANGTOOL-1514	Failed to process YANGs containing refine of a if-feature'd target	Resolved	10.0.0, 10.0.7, 7.0.18, 8.0.10, 9.0.0, 9.0.8	10.0.8, 11.0.0, 9.0.9
	YANGTOOL-1533	XmlParserStream reports "Unhandled mount-aware schema"	Resolved	10.0.9, 11.0.0, 3.0.17, 4.0.15, 5.0.10, 6.0.12, 7.0.18, 8.0.10, 9.0.8	10.0.10, 11.0.1, 9.0.9
	YANGTOOL-1537	CME in AbstractNodeContainerModificationStrategy.checkChildPreconditions()	Resolved	10.0.9, 11.0.1, 4.0.15, 5.0.10, 6.0.12, 7.0.18, 8.0.10, 9.0.8	10.0.10, 11.0.2, 9.0.9
	YANGTOOL-1543	XmlStringInstanceIdentifierCodec is using JSON encoding for writeValue()	Resolved	10.0.9, 11.0.2, 8.0.10, 9.0.8	10.0.10, 11.0.3, 9.0.9

Service Release Notes

Chlorine-SR1 Release Notes

This page details changes and bug fixes between the Chlorine Release and the Chlorine Stability Release 1 (Chlorine-SR1) of OpenDaylight.

daexim

- b42b03b : Bump upstreams
- 1495b43 : Bump MRI upstreams

integration/distribution

- ebe82ece : Add transportpce projects to distribution
- 10763de8 ODLPARENT-125 : Revert "Add transportpce projects in chlorine distribution"
- 54772930 : Bump upstreams
- 605c202b : Add transportpce projects in chlorine distribution
- d20ae873 : Bump bgpcep to 0.18.3
- ea45bd34 : Bump MRI upstreams
- 6647a6e2 : Bump version after release
- 34182893 : Enable TPCE in distribution

jsonrpc

- 50d36ac : Bump upstreams
- 1fe6a2b : Bump MRI upstreams

lispflowmapping

- 4f32857d : Bump upstreams
- 0cf875ee : Bump MRI upstreams

openflowplugin

- b96e66bed MDSAL-792 : Remove invalid test in OpenflowPortsUtilTest
- 291bf4ed6 : Bump upstreams
- 7a7d6b6c3 : Bump MRI upstreams

ovsdb

- 8d98fc586 : Bump upstreams
- 55e594919 : Bump MRI upstreams

serviceutils

- 8d5e578 : Bump upstreams
- 41f3bc4 : Bump upstreams

Chlorine-SR2 Release Notes

This page details changes and bug fixes between the Chlorine Stability Release 1 (Chlorine-SR1) and the Chlorine Stability Release 2 (Chlorine-SR2) of OpenDaylight.

daexim

- cf40729 : Bump upstreams

integration/distribution

- 3174ad63 : Bump upstreams

jsonrpc

- 372e3bf : Bump upstreams

lispflowmapping

- 95cdfdcf : Bump upstreams

openflowplugin

- 516e79904 : Bump upstreams

ovsdb

- 83c9ab332 : Bump upstreams

serviceutils

- db8816a : Bump upstreams

1.3 Getting Started Guide

1.3.1 Introduction

The OpenDaylight project is an open source platform for Software Defined Networking (SDN) that uses open protocols to provide centralized, programmatic control and network device monitoring.

Much as your operating system provides an interface for the devices that comprise your computer, OpenDaylight provides an interface that allows you to control and manage network devices.

What's different about OpenDaylight

Major distinctions of OpenDaylight's SDN compared to other SDN options are the following:

- A microservices architecture, in which a "microservice" is a particular protocol or service that a user wants to enable within their installation of the OpenDaylight controller, for example:
  - A plugin that provides connectivity to devices via the OpenFlow protocols (`openflowplugin`).
  - A platform service such as Authentication, Authorization, and Accounting (AAA).
  - A network service providing VM connectivity for OpenStack (netvirt).
- Support for a wide and growing range of network protocols: OpenFlow, P4 BGP, PCEP, LISP, NETCONF, OVSDB, SNMP and more.
- Model Driven Service Abstraction Layer (MD-SAL). Yang models play a key role in OpenDaylight and are used for:
  - Creating datastore schemata (tree based structure).
  - Generating application REST API (RESTCONF).
  - Automatic code generation (Java interfaces and Data Transfer Objects).

1.3.2 OpenDaylight concepts and tools

In this section we discuss some of the concepts and tools you encounter with basic use of OpenDaylight. The guide walks you through the installation process in a subsequent section, but for now familiarize yourself with the information below.

- To date, OpenDaylight developers have formed more than 50 projects to address ways to extend network functionality. The projects are a formal structure for developers from the community to meet, document release plans, code, and release the functionality they create in an OpenDaylight release.

  The typical OpenDaylight user will not join a project team, but you should know what projects are as we refer to their activities and the functionality they create. The Karaf features to install that functionality often share the project team's name.

- **Apache Karaf** provides a lightweight runtime to install the Karaf features you want to implement and is included in the OpenDaylight platform software. By default, OpenDaylight has no pre-installed features.

Features and feature repositories can be managed in the Karaf configuration file `etc/org.apache.karaf.features.cfg` using the `featuresRepositories` and `featuresBoot` variables.

- Model-Driven Service Abstraction Layer (MD-SAL) is the OpenDaylight framework that allows developers to create new Karaf features in the form of services and protocol drivers and connects them to one another. You can think of the MD-SAL as having the following two components:
  a. A shared datastore that maintains the following tree-based structures:
     i. The Config Datastore, which maintains a representation of the desired network state.
     ii. The Operational Datastore, which is a representation of the actual network state based on data from the managed network elements.
  b. A message bus that provides a way for the various services and protocol drivers to notify and communicate with one another.
- If you're interacting with OpenDaylight through the REST APIs while using the OpenDaylight interfaces, the microservices architecture allows you to select available services, protocols, and REST APIs.

1.3.3 Installing OpenDaylight

You complete the following steps to install your networking environment, with specific instructions provided in the subsections below.

Before detailing the instructions for these, we address the following: Java Runtime Environment (JRE) and operating system information Target environment Known issues and limitations

Install OpenDaylight

Downloading and installing OpenDaylight

The default distribution can be found on the OpenDaylight software download page: https://docs.opendaylight.org/en/latest/downloads.html

The Karaf distribution has no features enabled by default. However, all of the features are available to be installed.

Note: For compatibility reasons, you cannot enable all the features simultaneously. We try to document known incompatibilities in the *Install the Karaf features* section below.

Running the karaf distribution

To run the Karaf distribution:

1. Unzip the zip file.
2. Navigate to the directory.
3. run `./bin/karaf`.

For Example:

```
$ ls karaf-0.8.x-Oxygen.zip
karaf-0.8.x-Oxygen.zip
$ unzip karaf-0.8.x-Oxygen.zip
Archive:  karaf-0.8.x-Oxygen.zip
   creating: karaf-0.8.x-Oxygen/
   creating: karaf-0.8.x-Oxygen/configuration/
   creating: karaf-0.8.x-Oxygen/data/
   creating: karaf-0.8.x-Oxygen/data/tmp/
   creating: karaf-0.8.x-Oxygen/deploy/
   creating: karaf-0.8.x-Oxygen/etc/
   creating: karaf-0.8.x-Oxygen/externalapps/
   ...
   inflating: karaf-0.8.x-Oxygen/bin/start.bat
   inflating: karaf-0.8.x-Oxygen/bin/status.bat
   inflating: karaf-0.8.x-Oxygen/bin/stop.bat
$ cd distribution-karaf-0.8.x-Oxygen
$ ./bin/karaf

    ________                       ________                .__  .__       .__     __
    \_____  \ ______   ____   ____ \______ \ _____  ___.__.\|  \| \|__\| ____ \|  \|___/␣
↪ \|_
     /   \|   \\____ \_/ __ \ /    \ \|    \|  \\__  \<   \|  \|\|  \| \|  \|/ ___\\|  \
↪|  \   __\
    /    \|    \  \|_> >  ___/\|   \|  \\|    `   \/ __ \\___  \|\|  \|_\|  / /_/  >   Y␣
↪ \  \|
    \_______  /   __/ \___  >___\|  /_______  (____  / ____\|\|____/__\___  /\|___\|  /__
↪\|
            \/\|__\|        \/     \/        \/     \/\/           /_____/      \/
```

- Press `tab` for a list of available commands
- Typing `[cmd] --help` will show help for a specific command.
- Press `ctrl-d` or type `system:shutdown` or `logout` to shutdown OpenDaylight.

Note: Please take a look at the *Deployment Recommendations* and following sections under *Security Considerations* if you're planning on running OpenDaylight outside of an isolated test lab environment.

Install the Karaf features

To install a feature, use the following command, where feature1 is the feature name listed in the table below:

```
feature:install <feature1>
```

You can install multiple features using the following command:

```
feature:install <feature1> <feature2> ... <featureN-name>
```

Note: For compatibility reasons, you cannot enable all Karaf features simultaneously. The table below documents feature installation names and known incompatibilities.Compatibility values indicate the following:

- **all** - the feature can be run with other features.
- **self+all** - the feature can be installed with other features with a value of **all**, but may interact badly with other features that have a value of **self+all**. Not every combination has been tested.

Uninstalling features

To uninstall a feature, you must shut down OpenDaylight, delete the data directory, and start OpenDaylight up again.

Important: Uninstalling a feature using the Karaf feature:uninstall command is not supported and can cause unexpected and undesirable behavior.

Listing available features

To find the complete list of Karaf features, run the following command:

```
feature:list
```

To list the installed Karaf features, run the following command:

```
feature:list -i
```

The description of these features is in the project specific release notes Project-specific Release Notes section.

Karaf running on Windows 10

Windows 10 cannot be identify by Karaf (equinox). Issue occurs during installation of karaf features e.g.:

```
opendaylight-user@root>feature:install odl-restconf
Error executing command: Can't install feature odl-restconf/0.0.0:
Could not start bundle mvn:org.fusesource.leveldbjni/leveldbjni-all/1.8-odl in␣
↪feature(s) odl-akka-leveldb-0.7: The bundle "org.fusesource.leveldbjni.leveldbjni-all_
↪1.8.0 [300]" could not be resolved. Reason: No match found for native code: META-INF/
↪native/windows32/leveldbjni.dll; processor=x86; osname=Win32, META-INF/native/
↪windows64/leveldbjni.dll; processor=x86-64; osname=Win32, META-INF/native/osx/
↪libleveldbjni.jnilib; processor=x86; osname=macosx, META-INF/native/osx/libleveldbjni.
↪jnilib; processor=x86-64; osname=macosx, META-INF/native/linux32/libleveldbjni.so;␣
↪processor=x86; osname=Linux, META-INF/native/linux64/libleveldbjni.so; processor=x86-
↪64; osname=Linux, META-INF/native/sunos64/amd64/libleveldbjni.so; processor=x86-64;␣
↪osname=SunOS, META-INF/native/sunos64/sparcv9/libleveldbjni.so; processor=sparcv9;␣
↪osname=SunOS
```

Workaround is to add:

```
org.osgi.framework.os.name = Win32
```

to the karaf file:

```
etc/system.properties
```

The workaround and further info are in this thread: https://stackoverflow.com/questions/35679852/karaf-exception-is-thrown-while-installing-org-fusesource-leveldbjni

1.3.4 Setting Up Clustering

Clustering Overview

Clustering is a mechanism that enables multiple processes and programs to work together as one entity. For example, when you search for something on google.com, it may seem like your search request is processed by only one web server. In reality, your search request is processed by may web servers connected in a cluster. Similarly, you can have multiple instances of OpenDaylight working together as one entity.

Advantages of clustering are:

- Scaling: If you have multiple instances of OpenDaylight running, you can potentially do more work and store more data than you could with only one instance. You can also break up your data into smaller chunks (shards) and either distribute that data across the cluster or perform certain operations on certain members of the cluster.
- High Availability: If you have multiple instances of OpenDaylight running and one of them crashes, you will still have the other instances working and available.
- Data Persistence: You will not lose any data stored in OpenDaylight after a manual restart or a crash.

The following sections describe how to set up clustering on both individual and multiple OpenDaylight instances.

Multiple Node Clustering

The following sections describe how to set up multiple node clusters in OpenDaylight.

Deployment Considerations

To implement clustering, the deployment considerations are as follows:

- To set up a cluster with multiple nodes, we recommend that you use a minimum of three machines. You can set up a cluster with just two nodes. However, if one of the two nodes fail, the cluster will not be operational.

  Note: This is because clustering in OpenDaylight requires a majority of the nodes to be up and one node cannot be a majority of two nodes.

- Every device that belongs to a cluster needs to have an identifier. OpenDaylight uses the node's `role` for this purpose. After you define the first node's role as *member-1* in the `akka.conf` file, OpenDaylight uses *member-1* to identify that node.
- Data shards are used to contain all or a certain segment of a OpenDaylight's MD-SAL datastore. For example, one shard can contain all the inventory data while another shard contains all of the topology data.

  If you do not specify a module in the `modules.conf` file and do not specify a shard in `module-shards.conf`, then (by default) all the data is placed in the default shard (which must also be defined in `module-shards.conf` file). Each shard has replicas configured. You can specify the details of where the replicas reside in `module-shards.conf` file.
- If you have a three node cluster and would like to be able to tolerate any single node crashing, a replica of every defined data shard must be running on all three cluster nodes.

Note: This is because OpenDaylight's clustering implementation requires a majority of the defined shard replicas to be running in order to function. If you define data shard replicas on two of the cluster nodes and one of those nodes goes down, the corresponding data shards will not function.

- If you have a three node cluster and have defined replicas for a data shard on each of those nodes, that shard will still function even if only two of the cluster nodes are running. Note that if one of those remaining two nodes goes down, the shard will not be operational.
- It is recommended that you have multiple seed nodes configured. After a cluster member is started, it sends a message to all of its seed nodes. The cluster member then sends a join command to the first seed node that responds. If none of its seed nodes reply, the cluster member repeats this process until it successfully establishes a connection or it is shut down.
- After a node is unreachable, it remains down for configurable period of time (10 seconds, by default). Once a node goes down, you need to restart it so that it can rejoin the cluster. Once a restarted node joins a cluster, it will synchronize with the lead node automatically.

Clustering Scripts

OpenDaylight includes some scripts to help with the clustering configuration.

Note: Scripts are stored in the OpenDaylight `distribution/bin` folder, and maintained in the distribution project repository in the folder `karaf-scripts/src/main/assembly/bin/`.

Configure Cluster Script

This script is used to configure the cluster parameters (e.g. `akka.conf`, `module-shards.conf`) on a member of the controller cluster. The user should restart the node to apply the changes.

Note: The script can be used at any time, even before the controller is started for the first time.

Usage:

```
bin/configure_cluster.sh <index> <seed_nodes_list>
```

- index: Integer within 1..N, where N is the number of seed nodes. This indicates which controller node (1..N) is configured by the script.
- seed_nodes_list: List of seed nodes (IP address), separated by comma or space.

The IP address at the provided index should belong to the member executing the script. When running this script on multiple seed nodes, keep the seed_node_list the same, and vary the index from 1 through N.

Optionally, shards can be configured in a more granular way by modifying the file `"custom_shard_configs.txt"` in the same folder as this tool. Please see that file for more details.

Example:

```
bin/configure_cluster.sh 2 192.168.0.1 192.168.0.2 192.168.0.3
```

The above command will configure the member 2 (IP address 192.168.0.2) of a cluster made of 192.168.0.1 192.168.0.2 192.168.0.3.

Setting Up a Multiple Node Cluster

To run OpenDaylight in a three node cluster, perform the following:

First, determine the three machines that will make up the cluster. After that, do the following on each machine:

1. Copy the OpenDaylight distribution zip file to the machine.
2. Unzip the distribution.
3. Move into the <karaf-distribution-directory>/bin directory and run:

```
JAVA_MAX_MEM=4G JAVA_MAX_PERM_MEM=512m ./karaf
```

4. Enable clustering by running the following command at the Karaf command line:

```
feature:install odl-mdsal-distributed-datastore
```

   After installation you will be able to see new folder configuration/initial/ with config files

5. Open the following configuration files:
   - configuration/initial/akka.conf
   - configuration/initial/module-shards.conf
6. In each configuration file, make the following changes:

   Find every instance of the following lines and replace _127.0.0.1_ with the hostname or IP address of the machine on which this file resides and OpenDaylight will run:

```
artery {
  canonical.hostname = "127.0.0.1"
```

   Note: The value you need to specify will be different for each node in the cluster.

7. Find the following lines and replace _127.0.0.1_ with the hostname or IP address of any of the machines that will be part of the cluster:

```
cluster {
  seed-nodes = ["akka://opendaylight-cluster-data@${IP_OF_MEMBER1}:2550",
                <url-to-cluster-member-2>,
                <url-to-cluster-member-3>]
```

8. Find the following section and specify the role for each member node. Here we assign the first node with the *member-1* role, the second node with the *member-2* role, and the third node with the *member-3* role:

```
roles = [
  "member-1"
]
```

   Note: This step should use a different role on each node.

9. Open the configuration/initial/module-shards.conf file and update the replicas so that each shard is replicated to all three nodes:

```
replicas = [
    "member-1",
    "member-2",
    "member-3"
]
```

For reference, view a sample config files below.

10. Restart bundle via command line:

```
opendaylight-user@root>restart org.opendaylight.controller.sal-distributed-datastore
```

OpenDaylight should now be running in a three node cluster. You can use any of the three member nodes to access the data residing in the datastore.

Sample Config Files

Sample `akka.conf` file:

```
odl-cluster-data {
  akka {
    remote {
      artery {
        enabled = on
        transport = tcp
        canonical.hostname = "10.0.2.10"
        canonical.port = 2550
      }
    }

    cluster {
      # Using artery.
      seed-nodes = ["akka://opendaylight-cluster-data@10.0.2.10:2550",
                    "akka://opendaylight-cluster-data@10.0.2.11:2550",
                    "akka://opendaylight-cluster-data@10.0.2.12:2550"]

      roles = [
        "member-1"
      ]

      # when under load we might trip a false positive on the failure detector
      # failure-detector {
        # heartbeat-interval = 4 s
        # acceptable-heartbeat-pause = 16s
      # }
    }

    persistence {
      # By default the snapshots/journal directories live in KARAF_HOME. You can choose␣
↪to put it somewhere else by
      # modifying the following two properties. The directory location specified may be␣
↪a relative or absolute path.
      # The relative path is always relative to KARAF_HOME.
```

(continues on next page)

(continued from previous page)

```
      # snapshot-store.local.dir = "target/snapshots"

      # Use lz4 compression for LocalSnapshotStore snapshots
      snapshot-store.local.use-lz4-compression = false
      # Size of blocks for lz4 compression: 64KB, 256KB, 1MB or 4MB
      snapshot-store.local.lz4-blocksize = 256KB
    }
    disable-default-actor-system-quarantined-event-handling = "false"
  }
}
```

Sample `module-shards.conf` file:

```
module-shards = [
    {
        name = "default"
        shards = [
            {
                name="default"
                replicas = [
                    "member-1",
                    "member-2",
                    "member-3"
                ]
            }
        ]
    },
    {
        name = "topology"
        shards = [
            {
                name="topology"
                replicas = [
                    "member-1",
                    "member-2",
                    "member-3"
                ]
            }
        ]
    },
    {
        name = "inventory"
        shards = [
            {
                name="inventory"
                replicas = [
                    "member-1",
                    "member-2",
                    "member-3"
                ]
            }
```

(continues on next page)

(continued from previous page)

```
        ]
    },
    {
         name = "toaster"
         shards = [
             {
                 name="toaster"
                 replicas = [
                    "member-1",
                    "member-2",
                    "member-3"
                 ]
             }
         ]
    }
]
```

Cluster Monitoring

OpenDaylight exposes shard information via `MBeans`, which can be explored with `JConsole`, VisualVM, or other JMX clients, or exposed via a REST API using Jolokia, provided by the `odl-jolokia` Karaf feature. This is convenient, due to a significant focus on REST in OpenDaylight.

The basic URI that lists a schema of all available `MBeans`, but not their content itself is:

```
GET  /jolokia/list
```

To read the information about the shards local to the queried OpenDaylight instance use the following REST calls. For the config datastore:

```
GET  /jolokia/read/org.opendaylight.controller:type=DistributedConfigDatastore,
↪Category=ShardManager,name=shard-manager-config
```

For the operational datastore:

```
GET  /jolokia/read/org.opendaylight.controller:type=DistributedOperationalDatastore,
↪Category=ShardManager,name=shard-manager-operational
```

The output contains information on shards present on the node:

```
{
  "request": {
    "mbean": "org.opendaylight.controller:Category=ShardManager,name=shard-manager-
↪operational,type=DistributedOperationalDatastore",
    "type": "read"
  },
  "value": {
    "LocalShards": [
      "member-1-shard-default-operational",
      "member-1-shard-entity-ownership-operational",
      "member-1-shard-topology-operational",
      "member-1-shard-inventory-operational",
```

(continues on next page)

(continued from previous page)

```
      "member-1-shard-toaster-operational"
    ],
    "SyncStatus": true,
    "MemberName": "member-1"
  },
  "timestamp": 1483738005,
  "status": 200
}
```

The exact names from the "LocalShards" lists are needed for further exploration, as they will be used as part of the URI to look up detailed info on a particular shard. An example output for the `member-1-shard-default-operational` looks like this:

```
{
  "request": {
    "mbean": "org.opendaylight.controller:Category=Shards,name=member-1-shard-default-
↪operational,type=DistributedOperationalDatastore",
    "type": "read"
  },
  "value": {
    "ReadWriteTransactionCount": 0,
    "SnapshotIndex": 4,
    "InMemoryJournalLogSize": 1,
    "ReplicatedToAllIndex": 4,
    "Leader": "member-1-shard-default-operational",
    "LastIndex": 5,
    "RaftState": "Leader",
    "LastCommittedTransactionTime": "2017-01-06 13:19:00.135",
    "LastApplied": 5,
    "LastLeadershipChangeTime": "2017-01-06 13:18:37.605",
    "LastLogIndex": 5,
    "PeerAddresses": "member-3-shard-default-operational: akka://opendaylight-cluster-
↪data@192.168.16.3:2550/user/shardmanager-operational/member-3-shard-default-
↪operational, member-2-shard-default-operational: akka://opendaylight-cluster-data@192.
↪168.16.2:2550/user/shardmanager-operational/member-2-shard-default-operational",
    "WriteOnlyTransactionCount": 0,
    "FollowerInitialSyncStatus": false,
    "FollowerInfo": [
      {
        "timeSinceLastActivity": "00:00:00.320",
        "active": true,
        "matchIndex": 5,
        "voting": true,
        "id": "member-3-shard-default-operational",
        "nextIndex": 6
      },
      {
        "timeSinceLastActivity": "00:00:00.320",
        "active": true,
        "matchIndex": 5,
        "voting": true,
        "id": "member-2-shard-default-operational",
```

(continues on next page)

(continued from previous page)


```
        "nextIndex": 6
      }
    ],
    "FailedReadTransactionsCount": 0,
    "StatRetrievalTime": "810.5 s",
    "Voting": true,
    "CurrentTerm": 1,
    "LastTerm": 1,
    "FailedTransactionsCount": 0,
    "PendingTxCommitQueueSize": 0,
    "VotedFor": "member-1-shard-default-operational",
    "SnapshotCaptureInitiated": false,
    "CommittedTransactionsCount": 6,
    "TxCohortCacheSize": 0,
    "PeerVotingStates": "member-3-shard-default-operational: true, member-2-shard-
↪default-operational: true",
    "LastLogTerm": 1,
    "StatRetrievalError": null,
    "CommitIndex": 5,
    "SnapshotTerm": 1,
    "AbortTransactionsCount": 0,
    "ReadOnlyTransactionCount": 0,
    "ShardName": "member-1-shard-default-operational",
    "LeadershipChangeCount": 1,
    "InMemoryJournalDataSize": 450
  },
  "timestamp": 1483740350,
  "status": 200
}
```


The output helps identifying shard state (leader/follower, voting/non-voting), peers, follower details if the shard is a leader, and other statistics/counters.

The ODLTools team is maintaining a Python based tool, that takes advantage of the above `MBeans` exposed via `Jolokia`.

Failure handling

Overview

A fundamental problem in distributed systems is that network partitions (split brain scenarios) and machine crashes are indistinguishable for the observer, i.e. a node can observe that there is a problem with another node, but it cannot tell if it has crashed and will never be available again, if there is a network issue that might or might not heal again after a while or if process is unresponsive because of overload, CPU starvation or long garbage collection pauses.

When there is a crash, we would like to remove the affected node immediately from the cluster membership. When there is a network partition or unresponsive process we would like to wait for a while in the hope that it is a transient problem that will heal again, but at some point, we must give up and continue with the nodes on one side of the partition and shut down nodes on the other side. Also, certain features are not fully available during partitions so it might not matter that the partition is transient or not if it just takes too long. Those two goals are in conflict with each other and there is a trade-off between how quickly we can remove a crashed node and premature action on transient network partitions.

Split Brain Resolver

You need to enable the Split Brain Resolver by configuring it as downing provider in the configuration:

```
akka.cluster.downing-provider-class = "akka.cluster.sbr.SplitBrainResolverProvider"
```

You should also consider different downing strategies, described below.

Note: If no downing provider is specified, NoDowning provider is used.

All strategies are inactive until the cluster membership and the information about unreachable nodes have been stable for a certain time period. Continuously adding more nodes while there is a network partition does not influence this timeout, since the status of those nodes will not be changed to Up while there are unreachable nodes. Joining nodes are not counted in the logic of the strategies.

Setting `akka.cluster.split-brain-resolver.stable-after` to a shorter duration for having quicker removal of crashed nodes can be done at the price of risking a too early action on transient network partitions that otherwise would have healed. Do not set this to a shorter duration than the membership dissemination time in the cluster, which depends on the cluster size. Recommended minimum duration for different cluster sizes:

Cluster size	stable-after
5	7 s
10	10 s
20	13 s
50	17 s
100	20 s
1000	30 s

Note: It is important that you use the same configuration on all nodes.

When reachability observations by the failure detector are changed, the SBR decisions are deferred until there are no changes within the stable-after duration. If this continues for too long it might be an indication of an unstable system/network and it could result in delayed or conflicting decisions on separate sides of a network partition.

As a precaution for that scenario all nodes are downed if no decision is made within stable-after + down-all-when-unstable from the first unreachability event. The measurement is reset if all unreachable have been healed, downed or removed, or if there are no changes within stable-after * 2.

Configuration:

```
akka.cluster.split-brain-resolver {
  # Time margin after which shards or singletons that belonged to a downed/removed
  # partition are created in surviving partition. The purpose of this margin is that
  # in case of a network partition the persistent actors in the non-surviving partitions
  # must be stopped before corresponding persistent actors are started somewhere else.
  # This is useful if you implement downing strategies that handle network partitions,
  # e.g. by keeping the larger side of the partition and shutting down the smaller side.
  # Decision is taken by the strategy when there has been no membership or
  # reachability changes for this duration, i.e. the cluster state is stable.
  stable-after = 20s
```

(continues on next page)

(continued from previous page)

```
  # When reachability observations by the failure detector are changed the SBR decisions
  # are deferred until there are no changes within the 'stable-after' duration.
  # If this continues for too long it might be an indication of an unstable system/
↪network
  # and it could result in delayed or conflicting decisions on separate sides of a␣
↪network
  # partition.
  # As a precaution for that scenario all nodes are downed if no decision is made within
  # `stable-after + down-all-when-unstable` from the first unreachability event.
  # The measurement is reset if all unreachable have been healed, downed or removed, or
  # if there are no changes within `stable-after * 2`.
  # The value can be on, off, or a duration.
  # By default it is 'on' and then it is derived to be 3/4 of stable-after, but not less␣
↪than
  # 4 seconds.
  down-all-when-unstable = on
}
```

Keep majority

This strategy is used by default, because it works well for most systems. It will down the unreachable nodes if the current node is in the majority part based on the last known membership information. Otherwise down the reachable nodes, i.e. the own part. If the parts are of equal size the part containing the node with the lowest address is kept.

This strategy is a good choice when the number of nodes in the cluster change dynamically and you can therefore not use static-quorum.

- If there are membership changes at the same time as the network partition occurs, for example, the status of two members are changed to Up on one side but that information is not disseminated to the other side before the connection is broken, it will down all nodes on the side that could be in minority if the joining nodes were changed to Up on the other side. Note that if the joining nodes were not changed to Up and becoming a majority on the other side then each part will shut down itself, terminating the whole cluster.
- If there are more than two partitions and none is in majority each part will shut down itself, terminating the whole cluster.
- If more than half of the nodes crash at the same time the other running nodes will down themselves because they think that they are not in majority, and thereby the whole cluster is terminated.

The decision can be based on nodes with a configured role instead of all nodes in the cluster. This can be useful when some types of nodes are more valuable than others.

Configuration:

```
akka.cluster.split-brain-resolver.active-strategy=keep-majority
```

```
akka.cluster.split-brain-resolver.keep-majority {
  # if the 'role' is defined the decision is based only on members with that 'role'
  role = ""
}
```

Static quorum

The strategy named static-quorum will down the unreachable nodes if the number of remaining nodes are greater than or equal to a configured quorum-size. Otherwise, it will down the reachable nodes, i.e. it will shut down that side of the partition.

This strategy is a good choice when you have a fixed number of nodes in the cluster, or when you can define a fixed number of nodes with a certain role.

- If there are unreachable nodes when starting up the cluster, before reaching this limit, the cluster may shut itself down immediately. This is not an issue if you start all nodes at approximately the same time or use the `akka.cluster.min-nr-of-members` to define required number of members before the leader changes member status of 'Joining' members to 'Up'. You can tune the timeout after which downing decisions are made using the stable-after setting.
- You should not add more members to the cluster than quorum-size * 2 - 1. If the exceeded cluster size remains when a SBR decision is needed it will down all nodes because otherwise there is a risk that both sides may down each other and thereby form two separate clusters.
- If the cluster is split into 3 (or more) parts each part that is smaller than then configured quorum-size will down itself and possibly shutdown the whole cluster.
- If more nodes than the configured quorum-size crash at the same time the other running nodes will down themselves because they think that they are not in the majority, and thereby the whole cluster is terminated.

The decision can be based on nodes with a configured role instead of all nodes in the cluster. This can be useful when some types of nodes are more valuable than others.

By defining a role for a few stable nodes in the cluster and using that in the configuration of static-quorum you will be able to dynamically add and remove other nodes without this role and still have good decisions of what nodes to keep running and what nodes to shut down in the case of network partitions. The advantage of this approach compared to keep-majority is that you do not risk splitting the cluster into two separate clusters, i.e. a split brain.

Configuration:

```
akka.cluster.split-brain-resolver.active-strategy=static-quorum
```

```
akka.cluster.split-brain-resolver.static-quorum {
  # minimum number of nodes that the cluster must have
  quorum-size = undefined

  # if the 'role' is defined the decision is based only on members with that 'role'
  role = ""
}
```

Keep oldest

The strategy named keep-oldest will down the part that does not contain the oldest member. The oldest member is interesting because the active Cluster Singleton instance is running on the oldest member.

This strategy is good to use if you use Cluster Singleton and do not want to shut down the node where the singleton instance runs. If the oldest node crashes a new singleton instance will be started on the next oldest node.

- If down-if-alone is configured to on, then if the oldest node has partitioned from all other nodes the oldest will down itself and keep all other nodes running. The strategy will not down the single oldest node when it is the only remaining node in the cluster.

- If there are membership changes at the same time as the network partition occurs, for example, the status of the oldest member is changed to Exiting on one side but that information is not disseminated to the other side before the connection is broken, it will detect this situation and make the safe decision to down all nodes on the side that sees the oldest as Leaving. Note that this has the drawback that if the oldest was Leaving and not changed to Exiting then each part will shut down itself, terminating the whole cluster.

The decision can be based on nodes with a configured role instead of all nodes in the cluster.

Configuration:

```
akka.cluster.split-brain-resolver.active-strategy=keep-oldest
```

```
akka.cluster.split-brain-resolver.keep-oldest {
  # Enable downing of the oldest node when it is partitioned from all other nodes
  down-if-alone = on

  # if the 'role' is defined the decision is based only on members with that 'role',
  # i.e. using the oldest member (singleton) within the nodes with that role
  role = ""
}
```

Down all

The strategy named down-all will down all nodes.

This strategy can be a safe alternative if the network environment is highly unstable with unreachability observations that can't be fully trusted, and including frequent occurrences of indirectly connected nodes. Due to the instability there is an increased risk of different information on different sides of partitions and therefore the other strategies may result in conflicting decisions. In such environments it can be better to shutdown all nodes and start up a new fresh cluster.

- This strategy is not recommended for large clusters (> 10 nodes) because any minor problem will shutdown all nodes, and that is more likely to happen in larger clusters since there are more nodes that may fail.

Configuration:

```
akka.cluster.split-brain-resolver.active-strategy=down-all
```

Lease

The strategy named lease-majority is using a distributed lease (lock) to decide what nodes that are allowed to survive. Only one SBR instance can acquire the lease make the decision to remain up. The other side will not be able to acquire the lease and will therefore down itself.

This strategy is very safe since coordination is added by an external arbiter.

- In some cases the lease will be unavailable when needed for a decision from all SBR instances, e.g. because it is on another side of a network partition, and then all nodes will be downed.

Configuration:

```
akka {
  cluster {
    downing-provider-class = "akka.cluster.sbr.SplitBrainResolverProvider"
    split-brain-resolver {
```

(continues on next page)

(continued from previous page)

```
      active-strategy = "lease-majority"
      lease-majority {
        lease-implementation = "akka.coordination.lease.kubernetes"
      }
    }
  }
}
```

```
akka.cluster.split-brain-resolver.lease-majority {
  lease-implementation = ""

  # This delay is used on the minority side before trying to acquire the lease,
  # as an best effort to try to keep the majority side.
  acquire-lease-delay-for-minority = 2s

  # If the 'role' is defined the majority/minority is based only on members with that 'role
↪'.
  role = ""
}
```

Indirectly connected nodes

In a malfunctioning network there can be situations where nodes are observed as unreachable via some network links but they are still indirectly connected via other nodes, i.e. it's not a clean network partition (or node crash).

When this situation is detected the Split Brain Resolvers will keep fully connected nodes and down all the indirectly connected nodes.

If there is a combination of indirectly connected nodes and a clean network partition it will combine the above decision with the ordinary decision, e.g. keep majority, after excluding suspicious failure detection observations.

Multi-DC cluster

An OpenDaylight cluster has an ability to run on multiple data centers in a way, that tolerates network partitions among them.

Nodes can be assigned to group of nodes by setting the `akka.cluster.multi-data-center.self-data-center` configuration property. A node can only belong to one data center and if nothing is specified a node will belong to the default data center.

The grouping of nodes is not limited to the physical boundaries of data centers, it could also be used as a logical grouping for other reasons, such as isolation of certain nodes to improve stability or splitting up a large cluster into smaller groups of nodes for better scalability.

Failure detection

Failure detection is performed by sending heartbeat messages to detect if a node is unreachable. This is done more frequently and with more certainty among the nodes in the same data center than across data centers.

Two different failure detectors can be configured for these two purposes:

- `akka.cluster.failure-detector` for failure detection within own data center
- `akka.cluster.multi-data-center.failure-detector` for failure detection across different data centers

Heartbeat messages for failure detection across data centers are only performed between a number of the oldest nodes on each side. The number of nodes is configured with `akka.cluster.multi-data-center.cross-data-center-connections`.

This influences how rolling updates should be performed. Don't stop all of the oldest nodes that are used for gossip at the same time. Stop one or a few at a time so that new nodes can take over the responsibility. It's best to leave the oldest nodes until last.

Configuration:

```
multi-data-center {
  # Defines which data center this node belongs to. It is typically used to make islands␣
↪of the
  # cluster that are colocated. This can be used to make the cluster aware that it is␣
↪running
  # across multiple availability zones or regions. It can also be used for other logical
  # grouping of nodes.
  self-data-center = "default"


  # Try to limit the number of connections between data centers. Used for gossip and␣
↪heartbeating.
  # This will not limit connections created for the messaging of the application.
  # If the cluster does not span multiple data centers, this value has no effect.
  cross-data-center-connections = 5

  # The n oldest nodes in a data center will choose to gossip to another data center with
  # this probability. Must be a value between 0.0 and 1.0 where 0.0 means never, 1.0␣
↪means always.
  # When a data center is first started (nodes < 5) a higher probability is used so␣
↪other data
  # centers find out about the new nodes more quickly
  cross-data-center-gossip-probability = 0.2

  failure-detector {
    # FQCN of the failure detector implementation.
    # It must implement akka.remote.FailureDetector and have
    # a public constructor with a com.typesafe.config.Config and
    # akka.actor.EventStream parameter.
    implementation-class = "akka.remote.DeadlineFailureDetector"

    # Number of potentially lost/delayed heartbeats that will be
    # accepted before considering it to be an anomaly.
    # This margin is important to be able to survive sudden, occasional,
    # pauses in heartbeat arrivals, due to for example garbage collect or
```

(continues on next page)

(continued from previous page)

```
    # network drop.
    acceptable-heartbeat-pause = 10 s

    # How often keep-alive heartbeat messages should be sent to each connection.
    heartbeat-interval = 3 s

    # After the heartbeat request has been sent the first failure detection
    # will start after this period, even though no heartbeat message has
    # been received.
    expected-response-after = 1 s
  }
}
```

Active/Backup Setup

It is desirable to have the possibility to fail over to a different data center, in case all nodes become unreachable. To achieve that shards in the backup data center must be in "non-voting" state.

The API to manipulate voting states on shards is defined as RPCs in the cluster-admin.yang file in the *controller* project, which is well documented. A summary is provided below.

Note: Unless otherwise indicated, the below POST requests are to be sent to any single cluster node.

To create an active/backup setup with a 6 node cluster (3 active and 3 backup nodes in two locations) such configuration is used:

- for member-1, member-2 and member-3 (active data center):

```
akka.cluster.multi-data-center {
  self-data-center = "main"
}
```

- for member-4, member-5, member-6 (backup data center):

```
akka.cluster.multi-data-center {
  self-data-center = "backup"
}
```

There is an RPC to set voting states of all shards on a list of nodes to a given state:

```
POST  /restconf/operations/cluster-admin:change-member-voting-states-for-all-shards

or

POST  /rests/operations/cluster-admin:change-member-voting-states-for-all-shards
```

This RPC needs the list of nodes and the desired voting state as input. For creating the backup nodes, this example input can be used:

```
{
  "input": {
    "member-voting-state": [
```

(continues on next page)

(continued from previous page)

```
      {
        "member-name": "member-4",
        "voting": false
      },
      {
        "member-name": "member-5",
        "voting": false
      },
      {
        "member-name": "member-6",
        "voting": false
      }
    ]
  }
}
```

When an active/backup deployment already exists, with shards on the backup nodes in non-voting state, all that is needed for a fail-over from the active data center to backup data center is to flip the voting state of each shard (on each node, active AND backup). That can be easily achieved with the following RPC call (no parameters needed):

```
POST  /restconf/operations/cluster-admin:flip-member-voting-states-for-all-shards

or

POST /rests/operations/cluster-admin:flip-member-voting-states-for-all-shards
```

If it's an unplanned outage where the primary voting nodes are down, the "flip" RPC must be sent to a backup non-voting node. In this case there are no shard leaders to carry out the voting changes. However there is a special case whereby if the node that receives the RPC is non-voting and is to be changed to voting and there's no leader, it will apply the voting changes locally and attempt to become the leader. If successful, it persists the voting changes and replicates them to the remaining nodes.

When the primary site is fixed and you want to fail back to it, care must be taken when bringing the site back up. Because it was down when the voting states were flipped on the secondary, its persisted database won't contain those changes. If brought back up in that state, the nodes will think they're still voting. If the nodes have connectivity to the secondary site, they should follow the leader in the secondary site and sync with it. However if this does not happen then the primary site may elect its own leader thereby partitioning the 2 clusters, which can lead to undesirable results. Therefore it is recommended to either clean the databases (i.e., `journal` and `snapshots` directory) on the primary nodes before bringing them back up or restore them from a recent backup of the secondary site (see section *Backing Up and Restoring the Datastore*).

If is also possible to gracefully remove a node from a cluster, with the following RPC:

```
POST  /restconf/operations/cluster-admin:remove-all-shard-replicas

or

POST  /rests/operations/cluster-admin:remove-all-shard-replicas
```

and example input:

```
{
  "input": {
    "member-name": "member-1"
```

(continues on next page)

(continued from previous page)

```
  }
}
```

or just one particular shard:

```
POST  /restconf/operations/cluster-admin:remove-shard-replica

or

POST  /rests/operations/cluster-admin:remove-shard-replicas
```

with example input:

```
{
  "input": {
    "shard-name": "default",
    "member-name": "member-2",
    "data-store-type": "config"
  }
}
```

Now that a (potentially dead/unrecoverable) node was removed, another one can be added at runtime, without changing the configuration files of the healthy nodes (requiring reboot):

```
POST  /restconf/operations/cluster-admin:add-replicas-for-all-shards

or

POST  /rests/operations/cluster-admin:add-replicas-for-all-shards
```

No input required, but this RPC needs to be sent to the new node, to instruct it to replicate all shards from the cluster.

Note: While the cluster admin API allows adding and removing shards dynamically, the `module-shard.conf` and `modules.conf` files are still used on startup to define the initial configuration of shards. Modifications from the use of the API are not stored to those static files, but to the journal.

Extra Configuration Options

Name	Type	Default	Description
max-shard-data-change-executor-queue-size	uint32 (1..max)	1000	The maximum queue size for each shard's data store data change notification executor.
max-shard-data-change-executor-pool-size	uint32 (1..max)	20	The maximum thread pool size for each shard's data store data change notification executor.
max-shard-data-change-listener-queue-size	uint32 (1..max)	1000	The maximum queue size for each shard's data store data change listener.
max-shard-data-store-executor-queue-size	uint32 (1..max)	5000	The maximum queue size for each shard's data store executor.
shard-transaction-idle-timeout-in-minutes	uint32 (1..max)	10	The maximum amount of time a shard transaction can be idle without receiving any messages before it self-destructs.
shard-snapshot-batch-count	uint32 (1..max)	2000	The minimum number of entries to be present in the in-memory journal log before a snapshot is to be taken.
shard-snapshot-data-threshold-percentage	uint8 (1..100)	12	The percentage of `Runtime.totalMemory()` used by the in-memory journal log before a snapshot is to be taken
shard-heartbeat-interval-in-millis	uint16 (100..ma	500	The interval at which a shard will send a heart beat message to its remote shard.
operation-timeout-in-seconds	uint16 (5..max)	5	The maximum amount of time for akka operations (remote or local) to complete before failing.
shard-journal-recovery-log-batch-size	uint32 (1..max)	5000	The maximum number of journal log entries to batch on recovery for a shard before committing to the data store.
shard-transaction-commit-timeout-in-seconds	uint32 (1..max)	30	The maximum amount of time a shard transaction three-phase commit can be idle without receiving the next messages before it aborts the transaction
shard-transaction-commit-queue-capacity	uint32 (1..max)	2000	The maximum allowed capacity for each shard's transaction commit queue.
shard-initialization-timeout-in-seconds	uint32 (1..max)	300	The maximum amount of time to wait for a shard to initialize from persistence on startup before failing an operation (eg transaction create and change listener registration).
shard-leader-election-timeout-in-seconds	uint32 (1..max)	30	The maximum amount of time to wait for a shard to elect a leader before failing an operation (eg transaction create).
enable-metric-capture	boolean	false	Enable or disable metric capture.
bounded-mailbox-capacity	uint32 (1..max)	1000	Max queue size that an actor's mailbox can reach
persistent	boolean	true	Enable or disable data persistence
shard-isolated-leader-check-interval-in-millis	uint32 (1..max)	5000	the interval at which the leader of the shard will check if its majority followers are active and term itself as isolated

These configuration options are included in the `etc/org.opendaylight.controller.cluster.datastore.cfg` configuration file.

1.3.5 Persistence and Backup

Set Persistence Script

This script is used to enable or disable the config datastore persistence. The default state is enabled but there are cases where persistence may not be required or even desired. The user should restart the node to apply the changes.

Note: The script can be used at any time, even before the controller is started for the first time.

Usage:

```
bin/set_persistence.sh <on/off>
```

Example:

```
bin/set_persistence.sh off
```

The above command will disable the config datastore persistence.

Backing Up and Restoring the Datastore

The same cluster-admin API described in the *cluster guide* for managing shard voting states has an RPC allowing backup of the datastore in a single node, taking only the file name as a parameter:

```
POST  /restconf/operations/cluster-admin:backup-datastore

or

POST  /rests/operations/cluster-admin:backup-datastore
```

RPC input JSON:

```
{
  "input": {
    "file-path": "/tmp/datastore_backup"
  }
}
```

Note: This backup can only be restored if the YANG models of the backed-up data are identical in the backup OpenDaylight instance and restore target instance.

To restore the backup on the target node the file needs to be placed into the `$KARAF_HOME/clustered-datastore-restore` directory, and then the node restarted. If the directory does not exist (which is quite likely if this is a first-time restore) it needs to be created. On startup, ODL checks if the `journal` and `snapshots` directories in `$KARAF_HOME` are empty, and only then tries to read the contents of the `clustered-datastore-restore` directory, if it exists. So for a successful restore, those two directories should be empty. The backup file name itself does not matter, and the startup process will delete it after a successful restore.

The backup is node independent, so when restoring a 3 node cluster, it is best to restore it on each node for consistency. For example, if restoring on one node only, it can happen that the other two empty nodes form a majority and the cluster comes up with no data.

1.3.6 Security Considerations

This document discusses the various security issues that might affect OpenDaylight. The document also lists specific recommendations to mitigate security risks.

This document also contains information about the corrective steps you can take if you discover a security issue with OpenDaylight, and if necessary, contact the Security Response Team, which is tasked with identifying and resolving security threats.

Overview of OpenDaylight Security

There are many different kinds of security vulnerabilities that could affect an OpenDaylight deployment, but this guide focuses on those where (a) the servers, virtual machines or other devices running OpenDaylight have been properly physically (or virtually in the case of VMs) secured against untrusted individuals and (b) individuals who have access, either via remote logins or physically, will not attempt to attack or subvert the deployment intentionally or otherwise.

While those attack vectors are real, they are out of the scope of this document.

What remains in scope is attacks launched from a server, virtual machine, or device other than the one running OpenDaylight where the attack does not have valid credentials to access the OpenDaylight deployment.

The rest of this document gives specific recommendations for deploying OpenDaylight in a secure manner, but first we highlight some high-level security advantages of OpenDaylight.

- Separating the control and management planes from the data plane (both logically and, in many cases, physically) allows possible security threats to be forced into a smaller attack surface.
- Having centralized information and network control gives network administrators more visibility and control over the entire network, enabling them to make better decisions faster. At the same time, centralization of network control can be an advantage only if access to that control is secure.

  Note: While both previous advantages improve security, they also make an OpenDaylight deployment an attractive target for attack making understanding these security considerations even more important.

- The ability to more rapidly evolve southbound protocols and how they are used provides more and faster mechanisms to enact appropriate security mitigations and remediations.
- OpenDaylight is built from OSGi bundles and the Karaf Java container. Both Karaf and OSGi provide some level of isolation with explicit code boundaries, package imports, package exports, and other security-related features.
- OpenDaylight has a history of rapidly addressing known vulnerabilities and a well-defined process for reporting and dealing with them.

OpenDaylight Security Resources

- If you have any security issues, you can send a mail to *security@lists.opendaylight.org*.
- For the list of current OpenDaylight security issues that are either being fixed or resolved, refer to https://wiki-archive.opendaylight.org/view/Security:Advisories.
- To learn more about the OpenDaylight security issues policies and procedure, refer to https://wiki-archive.opendaylight.org/view/Security:Main

Deployment Recommendations

We recommend that you follow the deployment guidelines in setting up OpenDaylight to minimize security threats.

- The default credentials should be changed before deploying OpenDaylight.
- OpenDaylight should be deployed in a private network that cannot be accessed from the internet.
- Separate the data network (that connects devices using the network) from the management network (that connects the network devices to OpenDaylight).

Note: Deploying OpenDaylight on a separate, private management network does not eliminate threats, but only mitigates them. By construction, some messages must flow from the data network to the management network, e.g., OpenFlow *packet_in* messages, and these create an attack surface even if it is a small one.

- Implement an authentication policy for devices that connect to both the data and management network. These are the devices which bridge, likely untrusted, traffic from the data network to the management network.

Securing OSGi bundles

OSGi is a Java-specific framework that improves the way that Java classes interact within a single JVM. It provides an enhanced version of the *java.lang.SecurityManager* (ConditionalPermissionAdmin) in terms of security.

Java provides a security framework that allows a security policy to grant permissions, such as reading a file or opening a network connection, to specific code. The code maybe classes from the jarfile loaded from a specific URL, or a class signed by a specific key. OSGi builds on the standard Java security model to add the following features:

- A set of OSGi-specific permission types, such as one that grants the right to register an OSGi service or get an OSGi service from the service registry.
- The ability to dynamically modify permissions at run-time. This includes the ability to specify permissions by using code rather than a text configuration file.
- A flexible predicate-based approach to determining which rules are applicable to which *ProtectionDomain*. This approach is much more powerful than the standard Java security policy which can only grant rights based on a jarfile URL or class signature. A few standard predicates are provided, including selecting rules based upon bundle symbolic-name.
- Support for bundle *local permissions* policies with optional further constraints such as *DENY* operations. Most of this functionality is accessed by using the *OSGi ConditionalPermissionAdmin* service which is part of the OSGi core and can be obtained from the OSGi service registry. The *ConditionalPermissionAdmin* API replaces the earlier *PermissionAdmin* API.

For more information, refer to https://www.osgi.org

Securing the Karaf container

Apache Karaf is a OSGi-based run-time platform which provides a lightweight container for OpenDaylight and applications. Apache Karaf uses either Apache Felix Framework or Eclipse Equinox OSGi frameworks, and provide additional features on top of the framework.

Apache Karaf provides a security framework based on Java Authentication and Authorization Service (JAAS) in compliance with OSGi recommendations, while providing RBAC (Role-Based Access Control) mechanism for the console and Java Management Extensions (JMX).

The Apache Karaf security framework is used internally to control the access to the following components:

- OSGi services
- console commands
- JMX layer
- WebConsole

The remote management capabilities are present in Apache Karaf by default, however they can be disabled by using various configuration alterations. These configuration options may be applied to the OpenDaylight Karaf distribution.

Note: Refer to the following list of publications for more information on implementing security for the Karaf container.

- For role-based JMX administration, refer to https://karaf.apache.org/manual/latest/#_monitoring_and_management_using_jmx
- For remote SSH access configuration, refer to https://karaf.apache.org/manual/latest/#_remote
- For WebConsole access, refer to https://karaf.apache.org/manual/latest/#_webconsole
- For Karaf security features, refer to https://karaf.apache.org/manual/latest/#_security

Disabling the remote shutdown port

You can lock down your deployment post installation. Set `karaf.shutdown.port=-1` in `etc/custom.properties` or `etc/config.properties` to disable the remote shutdown port.

Securing Southbound Plugins

Many individual southbound plugins provide mechanisms to secure their communication with network devices. For example, the OpenFlow plugin supports TLS connections with bi-directional authentication and the NETCONF plugin supports connecting over SSH. Meanwhile, the Unified Secure Channel plugin provides a way to form secure, remote connections for supported devices.

When deploying OpenDaylight, you should carefully investigate the secure mechanisms to connect to devices using the relevant plugins.

Securing OpenDaylight using AAA

AAA stands for Authentication, Authorization, and Accounting. All three of these services can help improve the security posture of an OpenDaylight deployment.

The vast majority of OpenDaylight's northbound APIs (and all RESTCONF APIs) are protected by AAA by default when installing the +odl-restconf+ feature. In the cases that APIs are *not* protected by AAA, this will be noted in the per-project release notes.

By default, OpenDaylight has only one user account with the username and password *admin*. This should be changed before deploying OpenDaylight.

Securing RESTCONF using HTTPS

To secure Jetty RESTful services, including RESTCONF, you must configure the Jetty server to utilize SSL by performing the following steps.

1. Issue the following command sequence to create a self-signed certificate for use by the ODL deployment.

```
keytool -keystore .keystore -alias jetty -genkey -keyalg RSA
 Enter keystore password:  123456
What is your first and last name?
  [Unknown]:  odl
What is the name of your organizational unit?
  [Unknown]:  odl
What is the name of your organization?
  [Unknown]:  odl
What is the name of your City or Locality?
  [Unknown]:
What is the name of your State or Province?
  [Unknown]:
What is the two-letter country code for this unit?
  [Unknown]:
Is CN=odl, OU=odl, O=odl,
L=Unknown, ST=Unknown, C=Unknown correct?
  [no]:  yes
```

2. After the key has been obtained, make the following changes to the `etc/custom.properties` file to set a few default properties.

```
org.osgi.service.http.secure.enabled=true
org.osgi.service.http.port.secure=8443
org.ops4j.pax.web.ssl.keystore=./etc/.keystore
org.ops4j.pax.web.ssl.keystore.password=123456
org.ops4j.pax.web.ssl.keystore.type=PKCS12
org.ops4j.pax.web.ssl.key.password=123456
org.ops4j.pax.web.ssl.key.alias=jetty
```

3. Then edit the `etc/jetty.xml` file with the appropriate HTTP connectors.

   For example:

```
<?xml version="1.0"?>
<!--
 Licensed to the Apache Software Foundation (ASF) under one
```

(continues on next page)

(continued from previous page)

```
 or more contributor license agreements.  See the NOTICE file
 distributed with this work for additional information
 regarding copyright ownership.  The ASF licenses this file
 to you under the Apache License, Version 2.0 (the
 "License"); you may not use this file except in compliance
 with the License.  You may obtain a copy of the License at

   http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing,
software distributed under the License is distributed on an
"AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
 KIND, either express or implied.  See the License for the
 specific language governing permissions and limitations
 under the License.
-->
<!DOCTYPE Configure PUBLIC "-//Mort Bay Consulting//
DTD Configure//EN" "http://jetty.mortbay.org/configure.dtd">

<Configure id="Server" class="org.eclipse.jetty.server.Server">

    <!-- Use this connector for many frequently idle connections and for
        threadless continuations. -->
    <New id="http-default" class="org.eclipse.jetty.server.HttpConfiguration">
        <Set name="secureScheme">https</Set>
        <Set name="securePort">
            <Property name="jetty.secure.port" default="8443" />
        </Set>
        <Set name="outputBufferSize">32768</Set>
        <Set name="requestHeaderSize">8192</Set>
        <Set name="responseHeaderSize">8192</Set>

        <!-- Default security setting: do not leak our version -->
        <Set name="sendServerVersion">false</Set>

        <Set name="sendDateHeader">false</Set>
        <Set name="headerCacheSize">512</Set>
    </New>

    <Call name="addConnector">
        <Arg>
            <New class="org.eclipse.jetty.server.ServerConnector">
                <Arg name="server">
                    <Ref refid="Server" />
                </Arg>
                <Arg name="factories">
                    <Array type="org.eclipse.jetty.server.ConnectionFactory">
                        <Item>
                            <New class="org.eclipse.jetty.server.
↪HttpConnectionFactory">
                                <Arg name="config">
                                    <Ref refid="http-default"/>
```

(continues on next page)

(continued from previous page)

```
                                </Arg>
                            </New>
                        </Item>
                    </Array>
                </Arg>
                <Set name="host">
                    <Property name="jetty.host"/>
                </Set>
                <Set name="port">
                    <Property name="jetty.port" default="8181"/>
                </Set>
                <Set name="idleTimeout">
                    <Property name="http.timeout" default="300000"/>
                </Set>
                <Set name="name">jetty-default</Set>
            </New>
        </Arg>
    </Call>

    <!-- ============================================================= -->
    <!-- Configure Authentication Realms -->
    <!-- Realms may be configured for the entire server here, or -->
    <!-- they can be configured for a specific web app in a context -->
    <!-- configuration (see $(jetty.home)/contexts/test.xml for an -->
    <!-- example). -->
    <!-- ============================================================= -->
    <Call name="addBean">
        <Arg>
            <New class="org.eclipse.jetty.jaas.JAASLoginService">
                <Set name="name">karaf</Set>
                <Set name="loginModuleName">karaf</Set>
                <Set name="roleClassNames">
                    <Array type="java.lang.String">
                        <Item>org.apache.karaf.jaas.boot.principal.RolePrincipal</
↪Item>
                    </Array>
                </Set>
            </New>
        </Arg>
    </Call>
    <Call name="addBean">
        <Arg>
           <New class="org.eclipse.jetty.jaas.JAASLoginService">
                <Set name="name">default</Set>
                <Set name="loginModuleName">karaf</Set>
                <Set name="roleClassNames">
                    <Array type="java.lang.String">
                        <Item>org.apache.karaf.jaas.boot.principal.RolePrincipal</
↪Item>
                    </Array>
                </Set>
            </New>
```

(continues on next page)

(continued from previous page)

```
        </Arg>
    </Call>
</Configure>
```

The configuration snippet above adds a connector that is protected by SSL on port 8443. You can test that the changes have succeeded by restarting Karaf, issuing the following `curl` command, and ensuring that the 2XX HTTP status code appears in the returned message.

```
curl -u admin:admin -v -k https://localhost:8443/restconf/modules
```

Security Considerations for Clustering

While OpenDaylight clustering provides many benefits including high availability, scale-out performance, and data durability, it also opens a new attack surface in the form of the messages exchanged between the various instances of OpenDaylight in the cluster. In the current OpenDaylight release, these messages are neither encrypted nor authenticated meaning that anyone with access to the management network where OpenDaylight exchanges these clustering messages can forge and/or read the messages. This means that if clustering is enabled, it is even more important that the management network be kept secure from any untrusted entities.

1.3.7 What to Do with OpenDaylight

OpenDaylight (ODL) is a modular open platform for customizing and automating networks of any size and scale.

The following section provides links to documentation with examples of OpenDaylight deployment use cases.

Note: If you are an OpenDaylight contributor, we encourage you to add links to documentation with examples of interesting OpenDaylight deployment use cases in this section.

- OPNFV Installation instructions (Apex)
- Apex Wiki

1.3.8 How to Get Help

Users and developers can get support from the OpenDaylight community through the mailing lists, IRC and forums.

1. Create your question on ServerFault or Stackoverflow with the tag *#opendaylight*.

   Note: It is important to tag questions correctly to ensure that the questions reach individuals subscribed to the tag.

   ---

2. Mail `discuss@lists.opendaylight.org` or `dev@lists.opendaylight.org`.
3. Directly mail the PTL as indicated on the specific projects page.
4. IRC: Connect to `#opendaylight` or `#opendaylight-meeting` channel on `Libera.Chat`. The Linux Foundation's IRC guide may be helpful. You'll need an IRC client, or can use the Libera.Chat webchat, or perhaps you'll like IRCCloud.
5. For infrastructure and release engineering queries, mail `helpdesk@opendaylight.org`. IRC: Connect to `#lf-releng` channel on `Libera.Chat`.

1.4 Developer Guides

1.4.1 Generic

Developing apps on the OpenDaylight controller

This section provides the information required to develop apps on an OpenDaylight controller. Apps can be developed either within the controller using a Model-Driven SAL (MD-SAL) archetype or via an external app using the RESTCONF API to communicate with the controller.

Overview

This section starts app development within an OpenDaylight controller.

Perform the following steps to develop an app:

1. Create a local repository for the code using a simple build process.
2. Start the OpenDaylight controller.
3. Test a simple remote procedure call (RPC) that was created based on the principle of *hello world*.

Prerequisites

The following are the prerequisites for app creation:

- A development environment with the following setup and working correctly from the shell:
  - Maven 3.8.3 or later
  - Java 11-compliant JDK
  - An appropriate Maven settings.xml file. One way to get the default OpenDaylight settings.xml file is:

```
cp -n ~/.m2/settings.xml{,.orig}
wget -q -O - https://raw.githubusercontent.com/opendaylight/odlparent/master/
↪settings.xml > ~/.m2/settings.xml
```

Note: For Linux or Mac OS X development operating systems, the default local repository is ~/.m2/repository. For other platforms, the default local repository location varies.

Building an Example module

Perform the following steps to develop an app:

1. Create an *Example* project using Maven and an archetype called the *opendaylight-startup-archetype*. For first time downloads, this project will take some time to pull all the code from the remote repository.

```
mvn archetype:generate -DarchetypeGroupId=org.opendaylight.archetypes \
    -DarchetypeArtifactId=opendaylight-startup-archetype \
    -DarchetypeCatalog=remote -DarchetypeVersion=<VERSION>
```

The correct VERSION depends on desired Simultaneous Release:

Table 44: Archetype versions :widths: auto :header-rows: 1

OpenDaylight Simultaneous Release	`opendaylight-startup-archetype` version
Magnesium Development	1.3.0-SNAPSHOT
Aluminium Development	1.4.0-SNAPSHOT

2. Update the properties values. Ensure that the values for the `groupId` and the `artifactId` are in lower case.

```
Define value for property 'groupId': org.opendaylight.example
Define value for property 'artifactId': example
[INFO] Using property: version = 0.1.0-SNAPSHOT
Define value for property 'package' org.opendaylight.example: :
Define value for property 'classPrefix' Example: :
Define value for property 'copyright': Copyright (c) 2021 Yoyodyne, Inc.
[INFO] Using property: copyrightYear = 2021
```

3. Accept the default value of `classPrefix`, that is: `(${artifactId.substring(0,1).toUpperCase()}${artifactId.substring(1)})`. The `classPrefix` creates a Java Class Prefix by capitalizing the first character of the `artifactId`.

Note: This will create a directory with the name given to `artifactId` in the above dialog, with the following contents.

```
.gitreview
api/
artifacts/
cli/
features/
impl/
it/
karaf/
pom.xml
src/
```

4. Build the *example* project.

Note: Build time varies depending on the development machine specification. Ensure that you are in the project's root directory (example/,) and then issue the the following build command.

```
mvn clean install
```

5. Initialize the *example* project.

```
cd karaf/target/assembly/bin
ls
./karaf
```

6. Wait for the Karaf CLI to appear. Wait for OpenDaylight to fully load all components. This can take a minute or two after the prompt appears. Check the CPU on the dev machine, specifically the Java process to see when it slows down.

```
opendaylight-user@root>
```

7. Verify if the "example" module is built and search for the log entry that includes the entry *ExampleProvider Session Initiated*.

```
log:display | grep Example
```

8. Enter the following command to shutdown OpenDaylight through the console:

```
shutdown -f
```

Defining a simple HelloWorld RPC

1. Build a *hello* example from the Maven archetype *opendaylight-startup-archetype*, same as what was done in the previous steps.
2. View the entry point to understand the origins of the log line. The entry point starts in the `./impl` project:

```
impl/src/main/java/org/opendaylight/hello/impl/HelloProvider.java
```

3. Add any new content that you are doing in your implementation by using the `HelloProvider.init` method. It is analogous to an Activator.

```
/**
 * Method called when the blueprint container is created.
 */
public void init() {
    LOG.info("HelloProvider Session Initiated");
}
```

Add a simple HelloWorld RPC API

1. Navigate to `api/src/main/yang`.

```
cd api/src/main/yang/
```

2. Edit the `hello.yang` file. In the following example, we are adding the code in a YANG module to define the *hello-world* RPC:

```
module hello {
    yang-version 1.1;
    namespace "urn:opendaylight:params:xml:ns:yang:hello";
    prefix "hello";

    revision "2021-03-21" {
        description "Initial revision of hello model";
    }

    rpc hello-world {
        input {
            leaf name {
```

(continues on next page)

(continued from previous page)

```
                type string;
            }
        }
        output {
            leaf greeting {
                type string;
            }
        }
    }
}
```

3. Return to the `hello/api` directory. Do the following to build the API:

```
cd ../../../
mvn clean install
```

Implement the HelloWorld RPC API

1. Define the HelloService that was invoked through the *hello-world* API.

```
cd ../impl/src/main/java/org/opendaylight/hello/impl/
```

The `HelloProvider.java` file is in the current directory. Register the RPC that you created in the *hello.yang* file in the `HelloProvider.java` file. You can either edit the `HelloProvider.java` to match what is below or simply replace it with the code below.

```
/*
 * Copyright © 2021 Copyright (c) 2021 Yoyodyne, Inc. and others.  All rights␣
↪reserved.
 *
 * This program and the accompanying materials are made available under the
 * terms of the Eclipse Public License v1.0 which accompanies this distribution,
 * and is available at http://www.eclipse.org/legal/epl-v10.html
 */
package org.opendaylight.hello.impl;

import com.google.common.util.concurrent.ListenableFuture;
import org.opendaylight.mdsal.binding.api.DataBroker;
import org.opendaylight.yang.gen.v1.urn.opendaylight.params.xml.ns.yang.hello.
↪rev210321.HelloService;
import org.opendaylight.yang.gen.v1.urn.opendaylight.params.xml.ns.yang.hello.
↪rev210321.HelloWorldInput;
import org.opendaylight.yang.gen.v1.urn.opendaylight.params.xml.ns.yang.hello.
↪rev210321.HelloWorldOutput;
import org.opendaylight.yang.gen.v1.urn.opendaylight.params.xml.ns.yang.hello.
↪rev210321.HelloWorldOutputBuilder;
import org.opendaylight.yangtools.yang.common.RpcResult;
import org.opendaylight.yangtools.yang.common.RpcResultBuilder;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
```

(continues on next page)

(continued from previous page)

```
public class HelloProvider implements HelloService {

    private static final Logger LOG = LoggerFactory.getLogger(HelloProvider.class);

    private final DataBroker dataBroker;

    public HelloProvider(final DataBroker dataBroker) {
        this.dataBroker = dataBroker;
    }

    @Override
    public ListenableFuture<RpcResult<HelloWorldOutput>> helloWorld(HelloWorldInput␣
↪input) {
        HelloWorldOutputBuilder helloBuilder = new HelloWorldOutputBuilder();
        helloBuilder.setGreeting("Hello " + input.getName());
        return RpcResultBuilder.success(helloBuilder.build()).buildFuture();
    }

    /**
     * Method called when the blueprint container is created.
     */
    public void init() {
        LOG.info("HelloProvider Session Initiated");
    }

    /**
     * Method called when the blueprint container is destroyed.
     */
    public void close() {
        LOG.info("HelloProvider Closed");
    }
}
```

2. Update Blueprint XML file.

```
cd ../../../../../resources/OSGI-INF/blueprint/
```

You can either edit the `impl-blueprint.xml` to match what is below or simply replace it with the XML below.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- vi: set et smarttab sw=4 tabstop=4: -->
<!--
Copyright © 2021 Copyright (c) 2021 Yoyodyne, Inc. and others. All rights reserved.

This program and the accompanying materials are made available under the
terms of the Eclipse Public License v1.0 which accompanies this distribution,
and is available at http://www.eclipse.org/legal/epl-v10.html
-->
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
  xmlns:odl="http://opendaylight.org/xmlns/blueprint/v1.0.0"
  odl:use-default-for-reference-types="true">
```

(continues on next page)

(continued from previous page)

```
  <reference id="dataBroker"
    interface="org.opendaylight.mdsal.binding.api.DataBroker"
    odl:type="default" />

  <bean id="provider"
    class="org.opendaylight.hello.impl.HelloProvider"
    init-method="init" destroy-method="close">
    <argument ref="dataBroker" />
  </bean>

  <odl:rpc-implementation ref="provider"/>

</blueprint>
```

3. Optionally, users can build the Java classes that will register the new RPC. This is useful to test the edits that were made to `HelloProvider.java`.

```
cd ../../../../../
mvn clean install
```

4. Return to the top level directory

```
cd ../
```

5. Build the entire *hello* again. This will pickup the new changes, and then build them into the project:

```
mvn clean install
```

Execute the *hello* project for the first time

1. Run karaf

```
cd karaf/target/assembly/bin
./karaf
```

2. Wait for the project to load completely. Then view the log to see the loaded *Hello* Module:

```
log:display | grep Hello
```

Test the *hello-world* RPC via REST

There are a lot of ways to test a RPC. The following are a few examples.

1. Using the API Explorer through HTTP
2. Using a browser REST client

Using the API Explorer through HTTP

1. Navigate to apidoc UI with your web browser.

   Note: In the URL link for *apidoc UI*, change *localhost* to the IP/Host name to reflect your development machine's network address.

2. Enter the username and password. By default the credentials are *admin/admin*.
3. Select

```
hello
```

4. Select

```
POST /rests/operations/hello:hello-world
```

5. Click on the **Try it out** button.
6. Provide the required request input.

```
{
  "input": {
    "name": "Your Name"
  }
}
```

7. Select **application/json** for *Media type* in the *Responses* section.
8. Click the **Execute** button.
9. In the response body you should see

```
{
  "hello:output": {
    "greeting": "Hello Your Name"
  }
}
```

Using a browser REST client

Next, use a browser to POST a REST client request. For example, use the following information in the Firefox plugin:

RESTClient https://github.com/chao/RESTClient

```
POST: http://localhost:8181/rests/operations/hello:hello-world
```

Header:

```
Accept: application/json
Content-Type: application/json
Authorization: Basic admin admin
```

Body:

```
{
  "input": {
    "name": "Your Name"
  }
}
```

In the response body you should see:

```
{
  "hello:output": {
    "greeting": "Hello Your Name"
  }
}
```

Troubleshooting

If you get a response code 500 while attempting to `POST /rests/operations/hello:hello-world`, check the file: `impl/src/main/resources/OSGI-INF/blueprint/impl-blueprint.xml` and make sure the following element is specified for <blueprint>.

```
<odl:rpc-implementation ref="provider"/>
```

Integrating Animal Sniffer with OpenDaylight projects

This section provides information required to setup OpenDaylight projects with the Maven's Animal Sniffer plugin for testing API compatibility with OpenJDK.

Steps to setup up animal sniffer plugin with your project

1. Clone odlparent and checkout the required branch. The example below uses the branch 'origin/master/2.0.x'

```
git clone https://git.opendaylight.org/gerrit/odlparent
cd odlparent
git checkout origin/master/2.0.x
```

2. Modify the file *odlparent/pom.xml* to install the Animal Sniffer plugin as shown in the below example or refer to the change odlparent gerrit patch.

```
<plugin>
  <groupId>org.codehaus.mojo</groupId>
  <artifactId>animal-sniffer-maven-plugin</artifactId>
  <version>1.16</version>
  <configuration>
      <signature>
          <groupId>org.codehaus.mojo.signature</groupId>
          <artifactId>java18</artifactId>
          <version>1.0</version>
      </signature>
  </configuration>
  <executions>
```

(continues on next page)

(continued from previous page)

```
      <execution>
          <id>animal-sniffer</id>
          <phase>verify</phase>
          <goals>
              <goal>check</goal>
          </goals>
      </execution>
      <execution>
          <id>check-java-version</id>
          <phase>package</phase>
          <goals>
              <goal>build</goal>
          </goals>
          <configuration>
            <signature>
              <groupId>org.codehaus.mojo.signature</groupId>
              <artifactId>java18</artifactId>
              <version>1.0</version>
            </signature>
          </configuration>
      </execution>
  </executions>
</plugin>
```

3. Run a *mvn clean install* in odlparent.

```
mvn clean install
```

4. Clone the respective project to be tested with the plugin. As shown in the example in yangtools gerrit patch, modify the relevant pom.xml files to reference the version of odlparent which is checked-out. As shown in the example below change the version to *2.0.6-SNAPSHOT* or the version of the *2.0.x-SNAPSHOT* odlparent is checked out.

```
<parent>
    <groupId>org.opendaylight.odlparent</groupId>
    <artifactId>odlparent</artifactId>
    <version>2.0.6-SNAPSHOT</version>
    <relativePath/>
</parent>
```

5. Run a *mvn clean install* in your project.

```
mvn clean install
```

6. Run *mvn animal-sniffer:check* on your project and fix any relevant issues.

```
mvn animal-sniffer:check
```

Logging subsystem

Logging subsystem provides facilities for capturing, recording and reporting events which occur within the OpenDaylight system. These events are the primary means of system diagnostics and troubleshooting, serving a wide audience including automated monitoring systems, operators, administrators, support personnel and development engineers.

In order to provide a 'single system' experience, all software components should follow same basic rules for interfacing with the logging system. While it is not practical to force these rules on the various third parties, they should to be followed by all newly-developed components.

Logging in Java

Java provides a diverse set of logging APIs and implementations. OpenDaylight has selected SLF4J API, with the implementation being provided by the Karaf container.

The primary reasons for this decision are:

- proper split between API and implementation, allowing us to change the implementation without impacting code
- ability to provide legacy APIs (JUL, JCL, etc.) for third-party code

Message levels

SLF4J defines five levels of logging messages, ranging from **TRACE** to **ERROR**. The guidelines for their use are very vague and in the grand UNIX tradition mix severity of the message with its granularity. The following summary defines the basic rules on when to use a specific level, who the target audience is and how any message of the specified level will be interpreted.

Note that all events with level **INFO or higher present an API-like contract** of the system from the integration point-of-view: if they change, third-party systems such as monitoring may need to be updated to work correctly with the new system release. The message text on these levels should be understandable to people with networking and systems administration background, so any language assuming knowledge of programming in general, or Java in particular, should be avoided if at all possible.

Messages on **DEBUG** and higher present are part of interface contract with support entities, e.g. if they are changed operator and troubleshooting manuals, as well as knowledge-base systems may need to be updated to correctly interpret the information conveyed. On the **DEBUG** level, messages may assume a slight level of familiarity with general programming concepts. Terminology specific to any programming language should be avoided if possible.

ERROR

This level serves as the general error facility. It should be used whenever the software encounters an unexpected error which prevents further processing. The reporting component may attempt recovery which may affect system availability or performance for a period of time, or it may forcefully terminate if it is unable to recover. This level is always reported for all software components.

There are three examples when this level should be used:

1. Incorrigible internal state inconsistency, such as when a JVM reports an OutOfMemoryError. The top-level handler would log this condition and force a JVM exit, as it is not capable of continuing execution.
2. Internal state inconsistency, which is correctable by flushing and re-creating the state. In this case the component would log an event, which would indicate what assertion has been violated and that the state has been flushed to recover.

3. Request-level processing error, e.g. the application encounters an error which is preventing a particular request from completing, but there is no indication of systematic failure which would prevent other requests from being successfully processed.

The primary audience are monitoring systems and system operators, as the events reported have impact on operational performance of the system.

WARN

This level serves for events which indicate irregular circumstances, from which we have a clearly-defined recovery strategy which has no impact on system availability or performance as seen by the reporting component. Events on this level are usually reported.

A typical example for a candidate event is when a software component detects inconsistency within an external data feed and it performs a corrective action to compensate for it. Let's say we process a list of key/value pairs and encounter a duplicate key: we can either overwrite the old occurance, ignore the new occurance or abort. If we take any of the first two actions, we should report a **WARN** event. If we take the third, we should report an **ERROR** event.

The primary audience of these events are automated systems, operators and administrators, as this level of messages indicate non-optimal system operation (e.g. data feeds could use normalization) or may forebode a future failure.

INFO

This level serves for events which constitute major state changes within a software component – such as initialization, shutdown, persistent resource allocation, etc. – which are part of normal operations. Events on this level are typically reported for non-library components.

Each software component should log at least four events on this level:

1. when it starts its initialization
2. when it becomes operational,
3. when it starts orderly shutdown,
4. just before it terminates normally

The primary audience of these events are operators and administrators, who use them to confirm major interactions (such as restarting components) have occurred within the system.

DEBUG

This is the coarse diagnostic level, serving for events which indicate internal state transitions and detail what processing occurs. Events on this level are usually not reported, but are enabled when a subsystem code flows need to be examined for troubleshooting purposes.

Placement and amount of events generated at this level are at the discretion of the development engineers, as both relate directly to the component logic. The amount of detail in these events has to be limited to a single line of information useful for pinning down a misbehaving software component in a seemingly normal system and should be directly cross-referenceable to either previous **DEBUG** events or component configuration. A **hard requirement** is that there has to be at least one flow control statement (such as if, for, while) or a call to a different software component between two **DEBUG** event triggers.

Primary audience of these events are administrators and support personnel diagnosing operational issues, mainly in real-time as they occur on the system.

TRACE

This is the fine-grained diagnostic level, serving for events which indicate internal state transitions in full detail. Events on this level are not reported, but have to be explicitly enabled and may be collected for support purposes.

Placement, amount and contents of these events is completely at the description of development engineers. These events are completely release-specific, may change even between minor releases. Examples of events reported at this level would be method entry and exit, possibly including detailed input arguments, and dumps of internal data as it is being modified.

Primary audience of these events are senior support personnel and development engineers diagnosing operational irregularities which relate directly to code structure, mainly offline after being captured on a live system.

Logger instances

Each class containing calls to the logging subsystem **MUST** have its own logger, which is not shared with any other class. The variable holding reference to this logger **MUST** me named **'LOG'**. The easiest way to ensure you have the proper logger declaration is to use the following pattern:

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

class Foo {
    private static final Logger LOG = LoggerFactory.getLogger(Foo.class);
    ...
}
```

Use parameterized logging

Using dynamically-constructed message strings contributes to major overhead as the message string has to be constructed before the call to logging method is performed, thus forcing overhead even if the constructed string is not used (for example **DEBUG** level is not enabled).

Another issue with dynamically-constructed message strings is that they cannot be easily extracted by static source code analysis – a process critical for creating message catalogue of a particular software release, which in turn is needed for things like support knowledge bases, internationalization, etc.

While the former concern is addressed by Logger classes exposing methods such as `LOG.isDebugEnabled()`, the second concern can only be alleviated by using explicit String literals when calling the Logger methods. The correct way to address both concerns is to use parameterized logging as described at http://www.slf4j.org/faq.html#logging_performance. The basic pattern to follow is this:

```
class Foo {
    private static final Logger LOG = LoggerFactory.getLogger(Foo.class);

    // GOOD: string literal, no dynamic objects
    public void good_method(Object arg) {
        LOG.debug("Method called with arg {}", arg);
    }

    // BAD: string varies with argument
    public bad_method1(Object arg) {
        LOG.debug("Method called with arg " + arg);
```

(continues on next page)

(continued from previous page)

```
    }

    // BAD: code clutter
    public void bad_method2(Object arg) {
        if (LOG.isDebugEnabled()) {
            LOG.debug("Method called with arg {}", arg);
        }
    }

    // BAD: wrong level of language, this would be okay on TRACE
    public bad_method3(Object arg) {
        LOG.debug("arg is {}", arg);
    }
}
```

There is one thing that needs to be noted in this style, which is that logging an exception is properly supported if you supply it as the last argument, but you have to **MAKE SURE IT IS NOT HINTED TO IN THE MESSAGE STRING**:

```
class Foo {
    private static final Logger LOG = LoggerFactory.getLogger(Foo.class);

    // GOOD: note how there is no "{}" for ex
    public void good_method(Object arg) {
        try {
            doSomething(arg);
            ...
        } catch (SomeException ex) {
            LOG.warn("Failed to do something with {}, continuing", arg, ex);
        }
    }

    // BAD:
    // - exception is interpreted as an object
    // - exception chaining cause is lost
    // - stack trace is lost
    public void bad_method(Object arg) {
        try {
            doSomething(arg);
            ...
        } catch (SomeException ex) {
            LOG.warn("Failed to do something with {} because {}, continuing", arg, ex);
        }
    }
}
```

Avoid calls to the methods is{Trace|Debug|Info|Warn|Error}Enabled()

While it is true that methods such as `isDebugEnabled()` & Co. eliminate the minor overhead associated with the variadic method call, the burden on the developer is not acceptable simply because there are much better methods of automatic control of this overhead, without having any impact on the source code (or even the class files). One of them is JIT-level optimizations stemming from the ability to inline calls to `LOG.debug()`. The other is the set of interfaces from `java.lang.instrument` package, which can be used to completely eliminate the call overhead by removing all calls to `LOG.debug()` from the class bytecode based on the logger configuration.

The exception to this rule can be if you log something that has some cost to calculate - and if that log statement runs a lot. Something (made up) like:

```
for (int i = 0; i < 100000; i++) {
    if (LOG.isDebugEnabled()) {
        LOG.debug("The size is: {}", expensiveMethodToCalculateSize());
    }
}
```

Note that you can and always should pass Object and thus never `toString()` your objects passed to a Logger. For example, this is WRONG:

```
List<Interface> interfaces;
if (LOG.isDebugEnabled()) {
    LOG.info("Interfaces: {}", interfaces.toString());
}
```

and instead you can simply do:

```
LOG.info("Interfaces: {}", interfaces); // no need to guard this with isDebugEnabled!
```

Provide useful event context

Each logging call should provide useful context in which it occurred. This is not usually the case with a lot of Java-based software, notably even with some JVM implementations. Here are some typical anti-patterns which contribute to mitigated ability to diagnose problems when they happen:

```
class Foo {
    private static final Logger LOG = LoggerFactory.getLogger(Foo.class);

    // VERY BAD:
    // - no context provided
    // - non-constant message string
    // - assumes useful toString()
    public bad_method1(Object arg) {
        LOG.debug(arg.toString());
    }

    // VERY BAD:
    // - no context provided
    public bad_method2(Object arg) {
        LOG.debug("{}", arg);
    }
```

(continues on next page)

(continued from previous page)

```
    // COMPLETELY BAD:
    // - silently ignoring errors!!!
    public bad_method3(Object arg) {
        try {
            doSomething(arg);
            ...
        } catch (SomeException ex) {
        }
    }

    // EXTREMELY BAD:
    // - message is not constant
    // - no context is provided
    // - ex.getCause() is lost
    // - call stack is lost
    public void bad_method4(Object arg) {
        try {
            doSomething(arg);
            ...
        } catch (SomeException ex) {
            LOG.warn(ex.getMessage());
        }
    }

    // EXTREMELY BAD:
    // - message is not constant
    // - no context is provided
    // - ex.getCause() is probably lost
    // - call stack is probably lost
    // - assumes useful toString()
    public void bad_method5(Object arg) {
        try {
            doSomething(arg);
            ...
        } catch (SomeException ex) {
            LOG.warn(ex.toString());
        }
    }

    // VERY BAD:
    // - no useful context is provided
    // - ex.getCause() is probably lost
    // - call stack is probably lost
    // - administrators don't know what an Exception is!
    public void bad_method6(Object arg) {
        try {
            doSomething(arg);
            ...
        } catch (SomeException ex) {
            LOG.warn("Exception {}", ex);
        }
    }
```

(continues on next page)

(continued from previous page)

```
}
```

The proper fix for these anti-patterns is to always provide key information in the logging event:

- what went wrong
- how badly it went wrong
- in case we recover, shortly describe how (especially on **WARN** level)

```
class Foo {
    private static final Logger LOG = LoggerFactory.getLogger(Foo.class);

    // GOOD:
    // - string literal
    // - we explain what we tried to do
    // - we pass along information we have about the failure
    // - we explain that we recovered from the failure
    public void good_method1(Object arg) {
        try {
            doSomething(arg);
            ...
        } catch (SomeException ex) {
            LOG.warn("Failed to do something with {}, ignoring it", arg, ex);
        }
    }

    // GOOD:
    // - string literal
    // - we explain what we tried to do
    // - we pass along information we have about the failure
    // - we escalate the failure to our caller
    // - we also 'chain' the exception so it is not lost and can be
    // correlated
    public void good_method2(Object arg) {
        try {
            doSomething(arg);
            ...
        } catch (SomeException ex) {
            LOG.error("Failed to do something with {}", arg, ex);
            throw new RuntimeException("Failed to do something", ex);
        }
    }
}
```

Developer Test Guides

Generic

Build Time Testing with JUnit

Introduction to JUnit

JUnit is a framework for writing build time tests. See their home page http://junit.org/ for more information.

Build time tests are tests which run during the compilation phase of your project. Build time tests provide a way for a developer to quickly test (in isolation) the logic which belongs to a given project to verify it is behaving as desired.

Below we will provide the basics of JUnit tests and how you can use them, along with recommendations of how to write JUnit tests.

Additionally JUnit integrates nicely with the Eclipse and Idea frameworks to provide launching and debugging unit tests from within the IDE framework.

Qualities of a Good Unit Test

A good unit test will:

1. Run quickly
    - No sleeps or thread blocking if you can avoid it
2. Run against the public API's of a class (think interfaces!)
3. Test a "single" unit of work
    - Unit tests generally test an isolated piece of functionality. Integration tests are used to test the interaction between two pieces of logic.
    - You can have multiple tests in a single test class, each testing a different piece of functionality… more on that later.
4. Contain assertions which validate the functionality

To help get an idea if you tested all of your code you can use Code Coverage tools to review your code coverage. One such tools is EclEmma for eclipse which provides a mechanism to visually report the code coverage of or more unit tests.

Running a JUnit Test

Command Line

To run a JUnit tests via the command line you only need to run:

```
mvn test
```

However, that only runs the tests, it doesn't recompile them. In general running

```
mvn clean install
```

is the best way as it rebuilds your bundle and then executes any JUnit tests.

Eclipse

To run a JUnit tests via Eclipse, you can select the method, class, package, or source folder, right click it, and select *"Run As" -> "JUnit Test"*. The JUnit tests will now execute and an overview can bee seen in the JUnit view.

Debugging a Test

To debug a JUnit test via Eclipse, simply place your breakpoints as expected, and then right click the test method and choose *"Debug As" -> "JUnit Test"*. The test will start and will pause when the breakpoint is hit.

Example Implementation

Below we provide an example of how you can implement a JUnit test in your code.

Test Name and Package

A few things to keep in mind when creating a JUnit test class:

- Your JUnit test **MUST** start with, or end with the word **Test**. This is a keyword that the command line maven test invocation strategy looks for when deciding if it should inspect a class for JUnit annotations. If you class does not start or end with the word test it will NOT run via the maven command line and therefore will not run during builds either (however you will be able to invoke it via eclipse).
- It is best practice to place the JUnit test in the same package as the class that you are testing. This makes it easier to find the corresponding tests as well provides access to default members or methods if needed.
- You should try and name your class similar to the class under test (often just appending to the end the word "test" or if you have multiple test files, then appending a description of each file to the end of the file).

Adding JUnit to your Maven Project

To add the JUnit dependency to your maven project you simply add this dependency into the dependencies section of your pom file.

```
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.8.1</version>
  <scope>test</scope>
</dependency>
```

Note: You can exclude the scope and version tags assuming a parent project has defined the dependency in the dependency management section.

Creating your Test Class - Annotations

A JUnit test class is simply a java class. We use java annotations to mark the methods which set up the test and those which define actually tests.

In order to write a good unit test it is important to understand how JUnit executes your tests. JUnit will instantiate a new instance of your test class for each test method (@Test) that is defined in your test class. This means that non static class variables will be unique per instance of test method. This is done to ensure that one test method doesn't unintentionally interfere with another test in the same class. Unfortunately this also means that you need to initialize common intelligence every time which may lead to longer running tests (or messy code) which is something we want to avoid. JUnit makes this easier with a few additional annotations. Below is the flow of a JUnit test test:

1. **@BeforeClass** - This annotation can appear once in a test class on a public *static* method which takes no arguments. This provides a way to initialize some test data for ALL test methods in your test class. A BeforeClass method is intended to set up a static shared resource for all methods to use, such as a database connection or in our case a repository of yang model definitions (for example). In this method you can initialize static class variables for later use.
2. *A new instance of your test class is instantiated* - A new instance of your test class is not instantiated so each method can have its own sandbox.
3. **@Before** - This annotation can appear once in a test class on a public method which takes no arguments. This method is called before EVERY test method. It purpose to to provide a location for common initialization logic that each test method requires, such as instantiating the class under test and performing some generic set up. During this step you can class level variables for use in your test methods. Note, you do NOT have to worry about synchronization with the class level variables initialized here, because each test method has its own copy of the variables. Of course if your tests deals with concurrency, or you are modifying a static resource (not recommended) you may still need to deal with concurrency.
4. **@Test** - See above. At this point one of the @Test annotated methods is executed.
5. **@After** - The method with this annotation is executed after the @Test annotated method executes. This method is executed regardless of the passing or failing of the JUnit test and provides an opportunity to clean up after your test, such as deleting temporary files or deleting records from a database.
6. *At this point the testing framework will loop back to the @Before on a new test class instance to execute the next test method.*
7. **@AfterClass** - this annotation placed on a static method is used to clean up the initialization performed in the `@BeforeClass` method. It is executed once when all @Test methods have been executed.

Here is a sample JUnit test class that illustrates the ordering:

```
package org.opendaylight.controller.sal.restconf.impl.cnsn.to.json.test;

import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;

public class Temp {

   @BeforeClass
   public static void staticInit(){
       System.out.println( "Static Init" );
   }
```

(continues on next page)

(continued from previous page)

```
    @Before
    public void testInit(){
        System.out.println( "Test Init - " + this );
    }

    @Test
    public void testOne(){
        System.out.println( "Test One - " + this );
    }

    @Test
    public void testTwo(){
        System.out.println( "Test Two - " + this );
    }

    @After
    public void testCleanUp(){
        System.out.println( "Test Clean Up - " + this );
    }

    @AfterClass
    public static void staticCleanUp(){
        System.out.println( "Static Clean Up" );
    }

}
```

If you execute this through the test framework you would get output similar to this:

```
Static Init
Test Init - org.opendaylight.controller.sal.restconf.impl.cnsn.to.json.test.Temp@7476a6d9
Test One - org.opendaylight.controller.sal.restconf.impl.cnsn.to.json.test.Temp@7476a6d9
Test Clean Up - org.opendaylight.controller.sal.restconf.impl.cnsn.to.json.test.
↪Temp@7476a6d9
Test Init - org.opendaylight.controller.sal.restconf.impl.cnsn.to.json.test.Temp@7260c384
Test Two - org.opendaylight.controller.sal.restconf.impl.cnsn.to.json.test.Temp@7260c384
Test Clean Up - org.opendaylight.controller.sal.restconf.impl.cnsn.to.json.test.
↪Temp@7260c384
Static Clean Up
```

Notice that the object address is different for the two initialization calls, indicating that each test method did indeed receive its own object.

Note: It is important to not rely on the order of execution of the test methods when possible. JUnit can be configured to execute in parallel test methods from different classes but also inside the same class.

More details on parallel execution experimental support in Junit5 can be found at this URL:

https://junit.org/junit5/docs/current/user-guide/index.html#writing-tests-parallel-execution

---

Now that you have a shell for your test framework, its important to discuss how you indicate failures or passes in your JUnit tests. For that we need to discuss about assertions.

Validating Your Tests - Assertions

In order to have a complete unit test it is important to have quality assertions in your tests which actually validate that the behavior observed is the behavior that you wanted.

- **Passing a JUnit Test** - A JUnit test is considered to have passed if the method executes and returns without throwing an exception.
- **Failing a JUnit Test** - A JUnit test will be considered failed if an exception is thrown from it. There are a number of libraries that have been created to make this easier, most notable the Assert library.

The Assert library is closely tied to the JUnit library and provides methods to make it easy to validate that the data return is non null, equals another object etc. For example, if you want to assert that two objects are equal, you can use the `eassertEquals( ... )` method.

```
public void test(){
    Object expectedObj = ...
    Object actualObj = ...
    assertEquals( "Error message if not equal", expectedObj, actualObj );
}
```

This is a very common pattern for the assert methods. You can provide a string which provides a more descriptive error if things are not equal. Additionally, the Assertion framework will also to string the objects for comparison if things don't match to provide further information. Check out the Assert class for all of the other combinations: https://junit.org/junit4/javadoc/latest/org/junit/Assert.html

---

Note: It is a good idea to use the most appropriate method for your assertions. For example you can assert equality by simply doing `assertTrue( expectedObj.equals( actualObj ) )`. However the assert methods will do additional things like null checks, and printing out more detailed information on the error if the assertion does not pass. So in this case, using `assertEquals` is better as it would null check and print the values of the expected and actual objects for you automatically, making the act of asserting really easy!.

Concurrency in Unit test

In general, it is easier to avoid having multiple threads in your unit tests for a few reasons:

1. JUnit will only fail if an exception is thrown in the primary thread which it is executing your test from - exceptions thrown on other threads will not cause the test to fail!
2. Tests will slow down when other threads get spawned and you will start competing for system resources.
3. You have to deal with all of the other concurrency issues in your test that you do else where (waiting for threads, synchronizing objects etc) which makes the test harder to read.

If you do find that you need to deal with multiple threads in your test then you will need to take great care to make sure you are handling uncaught exceptions etc. If the class you are testing uses thread pools it is a good idea to refactor your test to pass in a ThreadPool instead of instantiate your own thread pool. If you do that, then you can use one of the following options to avoid multiple threads:

- Pass in a thread pool executor that executes the `runnable` / `callable` on the same thread
- Capture the `runnables` in a mock executer and then execute the run / call method at a later point.

[TODO - need to provide more examples for the above two cases]

Mocking

There are a number of frameworks out there which allow you to mock up objects in your Unit tests to simulate behavior. Most of these frameworks take advantage of good modular OO design (i.e. think interfaces, setters, getters etc). Some example mocking frameworks are:

- Mockito - https://github.com/mockito/mockito - **Note: Used in controller in a number of places**
- EasyMock - http://easymock.org/

Please refer to these sites for more details. If you have questions please reach out the mailing lists with questions - if there is enough interest we will develop more detailed best practices around mocking.

Considerations on Tests

Note: This document is a work in progress.

PowerMock

Avoid using PowerMock, if you can at all. It is black magic with special class loaders to mock static and what not really is a band aid for existing legacy code impossible to test. Consider refactoring the respective OpenDaylight code to be tested first, if you can.

Also think twice if you're using PowerMock on existing utility classes - you may be testing at the wrong level. For example, many projects have utilities that deals with the DataBroker (that's wrong, common shared utilities should be used). Many of these utility classes have static methods (that's wrong, they should get a DataBroker passed to their constructor, see `org.opendaylight.genius.datastoreutils.SingleTransactionDataBroker`, imagine there is no static there, that's again just for legacy). When writing a test, you could be tempted to use PowerMock on such static utility methods. That's wrong though - you should just let the code under test use such utilities - and use a test implementation of the DataBroker.

DataBroker

Typically you probably should not be mocking the DataBroker (using whatever framework), because there is a reasonably light weight test implementation of the entire persistence subsystem that is well suited for use in tests. You can easily use it by having your *Test class extends `org.opendaylight.controller.md.sal.binding.test.ConstantSchemaAbstractDataBrokerTest` and calling `getDataBroker()`, or just using the `org.opendaylight.controller.md.sal.binding.test.DataBrokerTestModule` static `dataBroker()` method.

NB that if you use the above, then (obviously) it makes no sense (anymore) to use `@RunWith(MockitoJUnitRunner.class)` and `@Mock DataBroker` and the transactions.

Mockito

Using Mockito can be fine - but keep an eye on how lines in your test are pure "Mockito infrastructure" related VS how many are "actually really test code". Do not "over-mock"; for example, there is absolutely no point in mocking implementations of the interfaces YANG code generations used e.g. for RPC input & output objects - just use the *Builder, like "real" code would. Also of course you would never Mock something like e.g. a Future or an Optional (seen in existing ODL code!).

Use "normal" Java, with a hint of Mockito

Be careful not to overstretch usage of Mockito and write complicated to read many lines of Mockito to set up a series of Mock objects if you could achieve the same relatively simply using standard Java, perhaps using a simple anonymous inner class. For example, the following is perfectly fine to do something like this in a test, and probably more readable to most people looking at your test than the equivalent in Mockito:

```
YourRpcService testYourRpcService = new YourRpcService() {
    @Override Future<RpcResult<Output>> someOperation(Input input) {
        if ( ... input some condition ...) {
            return Futures.immediateFailedFuture(RpcResultBuilder.failed());
        }
    };
    ...
}
```

If `YourRpcService` has other methods than just `someOperation()`, you can use a variant of "partial mocking":

```
import static org.opendaylight.yangtools.testutils.mockito.MoreAnswers.realOrException;

YourRpcService testYourRpcService = Mockito.mock(new YourRpcService() {
    @Override Future<RpcResult<Output>> someOperation(Input input) {
        if ( ... input some condition ...) {
            return Futures.immediateFailedFuture(RpcResultBuilder.failed());
        }
    }, realOrException());
    ...
}
```

Test Implementations of commonly used services

If several tests, or even other modules, are likely to use a stubbed service, then (only) it is worth to write something re-usable like this:

```
class abstract TestYourRpcService implements YourRpcService {
    public static YourRpcService newInstance() {
        return Mockito.mock(TestYourRpcService.class, realOrException());
    }
    @Override Future<RpcResult<Output>> someOperation(Input input) {
        ...
    }
    ...
}
```

Asserting Object structures

To assert the expected state of a tree of objects, typically but not necessarily data objects in Java objects generated by YANG binding, a number of projects use the vorburger/xtendbeans library.

The `org.opendaylight.mdsal.binding.testutils.AssertDataObjects` provides tight integration of this into OpenDaylight, including the `org.opendaylight.mdsal.binding.testutils.XtendBuilderExtensions`, which makes for a very readable syntax.

Component Tests

It's not that hard to write Component Tests which test the interaction of a number of interrelated services, without going to a full blown and much "heavier" Karaf OSGi integration test just yet; see *Component Tests (with Guice)*.

Integration Tests

Use Integration Tests (IT) to get a full Karaf OSGi runtime environment. *TODO Simplify that...*

Recommended Reading

- https://googletesting.blogspot.ch/2013/07/testing-on-toilet-know-your-test-doubles.html
- https://martinfowler.com/articles/mocksArentStubs.html

Component Tests (with Guice)

Note: This document is a work in progress.

Introduction

Code which uses dependency injection with standard annotations is well suite for component tests.

Component tests cover the functionality of 1 single OpenDaylight bundle (e.g. netvirt `aclservice`, genius `interfacemanager`, etc.) They are focused on end-to-end testing of APIs, not individual internal implementation classes (that's what Unit Tests are for). They focus on testing the code in their own module, and typically stub required external services. They assert outcome on the system, often those external services, and the data store. They wire together the internal beans through a Dependency Injection (DI) Framework called Guice, which leverages standard Java annotations in the code, which is equally used by Blueprints.

This presentation from the ODL Summit 2016 has some content related to this topic.

Maven

In order to use Google Guice in your end2end API component tests to wire bean objects using the same annotations as BP, use:

```
<dependency>
  <groupId>org.opendaylight.infrautils</groupId>
  <artifactId>inject.guice.testutils</artifactId>
  <version>${infrautils.version}</version>
  <scope>test</scope>
</dependency>
```

Code

Object Wiring Binding

You can write Guice object binding wiring classes like this:

```
public class AclServiceModule extends AbstractGuiceJsr250Module {
    @Override
    protected void configureBindings() {
        bind(AclServiceManager.class).to(AclServiceManagerImpl.class);
        bind(AclInterfaceStateListener.class);
    }
    ...
}
```

for any OSGi service external to the bundle (not local bean) you use bind() like this:

```
bind(DataTreeEventCallbackRegistrar).annotatedWith(OsgiService.class).
↪to(DataTreeEventCallbackRegistrarImpl.class)
```

JUnit with GuiceRule

Use *Module classes which define Object Wiring Binding in a JUnit * Test class like this:

```
public @Rule MethodRule guice = new GuiceRule(AclServiceModule.class,
↪ AclServiceTestModule.class);

@Inject DataBroker dataBroker;
@Inject AclServiceManager mdsalApiManager;
@Inject AclInterfaceStateListener mdsalApiManager;
```

Async

In these Component Tests (more so than in simple Unit Tests), one often hits problems due to the extensive use of highly asynchronous code in ODL. Some progress has been made with testing utilities for each respective async API, detailed in this chapter.

genius AsyncClusteredDataTreeChangeListenerBase & AsyncDataTreeChangeListenerBase

In order to make a test wait for something which happens in a AsyncClusteredDataTreeChangeListenerBase or AsyncDataTreeChangeListenerBase subclass before then asserting on the outcome of what happened, you just add the TestableDataTreeChangeListenerModule to the GuiceRule of your *Test, and then `@Inject AsyncEventsWaiter asyncEventsWaiter`, and use `awaitEventsConsumption()` AFTER having done action like a data store write for which a listener should kick in, and BEFORE reading the datastore to check the effect:

```
public final @Rule MethodRule guice = new GuiceRule(
    new YourTestModule(),
    new TestableDataTreeChangeListenerModule());
@Inject AsyncEventsWaiter asyncEventsWaiter;
asyncEventsWaiter.awaitEventsConsumption();
```

If a AsyncClusteredDataTreeChangeListenerBase or AsyncDataTreeChangeListenerBase (subclass) has "fired", then the AsyncEventsWaiter verifies that a test has indeed used `awaitEventsConsumption()` - and fails the test with IllegalStateException: Test forgot an `awaitEventsConsumption()` if it does not. This mechanism ensures that a test does not "forget" to `awaitEventsConsumption` and assert an expected outcome. NB however that if the test runs fast, it may end before the listeners kicked in, and the IllegalStateException may not always been seen (i.e. leading to a `"heisenbug"`, found with the RunUntilFailureRule). Therefore, if in your test you do not need to `awaitEventsConsumption()` at all, then you should not use the TestableDataTreeChangeListenerModule. However, this is likely an indication of lack of better test coverage in your test - you probably do want to assert on the effect of your AsyncClusteredDataTreeChangeListenerBase or AsyncDataTreeChangeListenerBase subclasses?

infrautils JobCoordinator (formerly genius DataStoreJobCoordinator)

similarly to above, using the JobCoordinatorEventsWaiter:

```
@Inject JobCoordinatorEventsWaiter coordinatorEventsWaiter;
coordinatorEventsWaiter.awaitEventsConsumption();
(TODO still need to be ported from genius to infrautils)
(TODO need to write a combined AsyncEventsWaiter instead of doing e.g.␣
↪InterfaceManagerTestUtil's waitTillOperationCompletes)
```

It is HIGHLY (!) recommended to FIRST switch code from the @Deprecated DataStoreJobCoordinator (in genius) to the JobCoordinator (in infrautils), because that does not suffer from the problem where a background job can "continue on" from one @Test method into another @Test, or even from one *Test class into another, due to use of "static", which can lead to VERY confusing log messages.

genius ResourceBatchingManager

The ResourceBatchingManager API does not yet have an AsyncEventsWaiter companion.

Other

Some of our "new style" Component Tests, such as e.g. InterfaceManagerConfigurationTest, and others, still need Thread.sleep() in some places.. the eventual goal is to be able to eventually completely eliminate them from all tests.

Tutorial

Let's imagine you want to make a change e.g. in `aclservice`, just as an example. Specifically, you've added a new argument for another new internal bean or external service to the @Singleton AclServiceManagerImpl @Inject annotated constructor, let's say to an IdManagerService for the sake of this example discussion.

A component test based on Guice wiring, such as AclServiceTest, will now fail on you with a message saying something like this:

No implementation for (... your new service...) was bound while locating (...) for the X-th parameter of AclServiceManagerImpl.

The *Module classes referenced from the GuiceRule in a *Test is where the wiring is defined - that's what determines, for that test, what implementation class is bound to what service interface etc. If you have a look at e.g. the AclServiceModule & AclServiceTestModule, it should be obvious what that does - just 1 single line for each binding.

The error message shown above simply means that an interface was encountered but you have not specified what implementation you would like to use for that interface in a given test. (Different tests could have different Module with varying bindings; but don't have to.)

To fix this after having made your change, you would now have to add 1 line in AclServiceTestModule to do a bind() of IdManagerService to... something.

If IdManagerServiceas was some new internal helper class of `aclservice` which you would like to test, then you would just do:

```
bind(IdManagerService.class).to(YOURIdManagerServiceImpl.class);
```

The YOURIdManagerServiceImpl would have a @Singleton annotation on its class, and have an @Inject annotation on its constructor, to automatically get its dependencies injected (and perhaps have `@PostConstruct` and `@PreDestroy`, if it has a `"lifecycle"`; or extend AbstractLifecycle). This is further documented on the DI Guidelines page.

Now, in the case of an existing ODL service from another project, you typically didn't actually write your own implementation of the IdManagerService interface. At full system runtime, you probably would like that to use the IdManager class (and you probably added that to your BP XML). So, having understood above, you COULD now be tempted to add this in AclServiceTestModule:

```
bind(IdManagerService.class).to(IdManager.class);
```

but there is two problems with this, 1. small practical (easy to fix), 2. conceptual (more important):

1. IdManager at the time of initially writing this documentation did not have `@Singleton` `@Inject` and `@PreDestroy` on its `close()` .. this may have already changed - or you could, easily, make a contribution to Genius to change that; I would recommend making IdManager extend AbstractLifecycle in this case. This can theoretically, even though we wouldn't recommend that, also be worked-around by doing the IdManager "wiring" manually through 2 lines of like new `IdManager(...)` and then use `bind(IdManagerService.class) .toInstance(myIdManager)`. BUT...

2. ... it would, typically IMHO, be wrong to use IdManager as IdManagerService implementation in AclServiceTest. This is more of a general recommendation than a hard rule. The idea is that the component test of `aclservice` should NOT have to depend on the real implementation of all external services the `aclservice` code depends on (only all of the internal beans of `aclservice`). So it would, generally, be considered better to bind a local test implementation of IdManager, which does the minimum you need for the test. A full coverage test of IdManager would be the responsibility of genius `idmanager-impl`, not `aclservice-impl`. So what I would probably start with doing in your case, unless there is a very strong need that you must absolutely have the "full" IdManager for the AclServiceTest, is to just put this into AclServiceTestModule's configure() method:

```
// import static org.opendaylight.yangtools.testutils.mockito.MoreAnswers.*;
bind(IdManagerService.class).toInstance(Mockito.mock(IdManagerService.class,
↪ exception());
```

Doing this will resolve the Guice Exception you have run into below. But whenever some `aclservice` code now actually calls a method IdManagerService, you'll get an UnstubbedMethodException - and this is normal - because you just mocked IdManagerService! I would still recommend to start like this, and then go about fixing UnstubbedMethodException as they arise when you run AclServiceTest ...

Let us for example say that your new code calls IdManagerServices `allocateIdRange()` method somewhere - I don't know if it does, so this is just for Illustration. You could make your mocked IdManagerService do something else than throw a UnstubbedMethodException for `allocateIdRange()` in two different "styles", this is somewhat dependant on personal preference:

A) Write out a partial "fake" implementation of it:

Write an inner class right there inside at the end of the AclServiceTestModule.java - just because it's easier to have this together and immediately evident when reading code; unless it becomes very long, in which case you could also move it outside, of course:

```
private abstract static class TestIdManagerService implements IdManagerService {
    @Override
    public Future<RpcResult<AllocateIdOutput>> allocateId(AllocateIdInput input) {
    // TODO do something minimalistic here, just useful for the test, not a general␣
↪implementation
    }
}
```

Note that the code in such test service implementations are typically simplistic and trivial, and not "real full fledged". Note also that only methods which the test actually requires are implemented; because it's abstract, we don't have to write anything at all for other methods of the interface.

You can then change the binding in configure() to be:

```
bind(IdManagerService.class).toInstance(Mockito.mock(TestIdManagerService.class,
↪ realOrException())
```

Note the subtle difference with the use of `realOrException()` instead of just `exception()`.

This first style is Vorburger's personal preference; finding this code clearer to read and understand for anyone than "traditional" Mockito usage, and not minding to have to type a few extra lines (for the class), which the IDE will put for me on `Ctrl-Space` anyway, than having to understand the Mockito magic. This is particular true when the implemented methods have anything but non-trivial arguments and return types - which is often the case in ODL.

B) Write the implementation using traditional Mockito API:

Write a method, just for clarify, such as:

```
private IdManagerService idManagerService() {
    IdManagerService idManagerService = Mockito.mock(IdManagerService.class);
    Mockito.when(idManagerService.allocateId(...)).thenReturn(...);
    // etc.
    return idManagerService;
}
```

and then changing the binding in configure() to be:

```
bind(IdManagerService.class).toInstance(idManagerService());
```

1.4.2 Core features Developer Guides

Distribution Version reporting

Overview

This section provides an overview of **odl-distribution-version** feature.

A remote user of OpenDaylight usually has access to RESTCONF and NETCONF northbound interfaces, but does not have access to the system OpenDaylight is running on. OpenDaylight has released multiple versions including Service Releases, and there are incompatible changes between them. In order to know which YANG modules to use, which bugs to expect and which workarounds to apply, such user would need to know the exact version of at least one OpenDaylight component.

There are indirect ways to deduce such version, but the direct way is enabled by `odl-distribution-version` feature. Administrator can specify version strings, which would be available to users via NETCONF, or via RESTCONF if OpenDaylight is configured to initiate NETCONF connection to its config subsystem northbound interface.

By default, users have write access to config subsystem, so they can add, modify or delete any version strings present there. Admins can only influence whether the feature is installed, and initial values.

Config subsystem is local only, not cluster aware, so each member reports versions independently. This is suitable for heterogeneous clusters. On homogeneous clusters, make sure you set and check every member.

Key APIs and Interfaces

Current implementation relies heavily on `config-parent` parent POM file from Controller project.

YANG model for config subsystem

Throughout this chapter, *model* denotes YANG module, and *module* denotes item in config subsystem module list.

Version functionality relies on config subsystem and its `config` YANG model. The YANG model `odl-distribution-version` adds an identity `odl-version` and augments `/config:modules/module/configuration` adding new case for `odl-version` type. This case contains single leaf `version`, which would hold the version string.

Config subsystem can hold multiple modules, the version string should contain version of OpenDaylight component corresponding to the module name. As this is pure metadata with no consequence on OpenDaylight behavior, there is no prescribed scheme for choosing config module names. But see the default configuration file for examples.

Java API

Each config module needs to come with java classes which override `customValidation()` and `createInstance()`. Version related modules have no impact on OpenDaylight internal behavior, so the methods return void and dummy closeable respectively, without any side effect.

Default config file

Initial version values are set via config file `odl-version.xml` which is created in `$KARAF_HOME/etc/opendaylight/karaf/` upon installation of `odl-distribution-version` feature. If admin wants to use different content, the file with desired content has to be created there before feature installation happens.

By default, the config file defines two config modules, named `odl-distribution-version` and `odl-odlparent-version`.

Currently the default version values are set to Maven property strings (as opposed to valid values), as the needed new functionality did not make it into Controller project in Boron. See Bug number 6003.

Karaf Feature

The `odl-distribution-version` feature is currently the only feature defined in feature repository of `artifactId features-distribution`, which is available (transitively) in OpenDaylight Karaf distribution.

RESTCONF usage

OpenDaylight config subsystem NETCONF northbound is not made available just by installing `odl-distribution-version`, but most other feature installations would enable it. RESTCONF interfaces are enabled by installing `odl-restconf` feature, but that do not allow access to config subsystem by itself.

On single node deployments, installation of `odl-netconf-connector-ssh` is recommended, which would configure `controller-config` device and its MD-SAL mount point. See documentation for clustering on how to create similar devices for member modes, as `controller-config` name is not unique in that context.

Assuming single node deployment and user located on the same system, here is an example `curl` command accessing `odl-odlparent-version` config module:

```
curl 127.0.0.1:8181/restconf/config/network-topology:network-topology/topology/topology-
↪netconf/node/controller-config/yang-ext:mount/config:modules/module/odl-distribution-
↪version:odl-version/odl-odlparent-version
```

Distribution features

Overview

This section provides an overview of **odl-integration-compatible-with-all** and **odl-integration-all** features.

Integration/Distribution project produces a Karaf 4 distribution which gives users access to many Karaf features provided by upstream OpenDaylight projects. Users are free to install arbitrary subset of those features, but not every feature combination is expected to work properly.

Some features are pro-active, which means OpenDaylight in contact with other network elements starts diving changes in the network even without prompting by users, in order to satisfy initial conditions their use case expects. Such activity from one feature may in turn affect behavior of another feature.

In some cases, there exists features which offer different implementation of the same service, they may fail to initialize properly (e.g. failing to bind a port already bound by the other feature).

Integration/Test project is maintaining system tests (CSIT) jobs. Aside of testing scenarios with only a minimal set of features installed (-only- jobs), the scenarios are also tested with a large set of features installed (-all- jobs).

In order to define a proper set of features to test with, Integration/Distribution project defines two "aggregate" features. Note that these features are not intended for production use, so the feature repository which defines them is not enabled by default.

The content of these features is determined by upstream OpenDaylight contributions, with Integration/Test providing insight on observed compatibility relations. Integration/Distribution team is focused only on making sure the build process is reliable.

Feature repositories

features-index

This feature repository is enabled by default. It does not refer to any new features directly, instead it refers to upstream feature repositories, enabling any feature contained therein to be available for installation.

features-test

This feature repository defines the two aggregate features. To enable this repository, change the `featuresRepositories` line of `org.apache.karaf.features.cfg` file, by copy-pasting the feature-index value and editing the name.

Karaf features

The two aggregate features, defining sets of user-facing features defined by compatibility requirements. Note that is the compatibility relation differs between single node an cluster deployments, single node point of view takes precedence.

`odl-integration-all`

This feature contains the largest set of user-facing features which may affect each others operation, but the set does not affect usability of Karaf infrastructure.

Note that port binding conflicts and "server is unhealthy" status of config subsystem are considered to affect usability, as is a failure of RESTCONF to respond to GET on `/restconf/modules` with HTTP status 200.

This feature is used in verification process for Integration/Distribution contributions.

odl-integration-compatible-with-all

This feature contains the largest set of user-facing features which are not pro-active and do not affect each others operation.

Installing this set together with just one of `odl-integration-all` features should still result in fully operational installation, as one pro-active feature should not lead to any conflicts. This should also hold if the single added feature is outside `odl-integration-all`, if it is one of conflicting implementations (and no such implementations is in `odl-integration-all`).

This feature is used in the aforementioned -all- CSIT jobs.

Neutron Service Developer Guide

Overview

This Karaf feature (`odl-neutron-service`) provides integration support for OpenStack Neutron via the OpenDaylight ML2 mechanism driver. The Neutron Service is only one of the components necessary for OpenStack integration. It defines YANG models for OpenStack Neutron data models and northbound API via REST API and YANG model RESTCONF.

Those developers who want to add new provider for new OpenStack Neutron extensions/services (Neutron constantly adds new extensions/services and OpenDaylight will keep up with those new things) need to communicate with this Neutron Service or add models to Neutron Service. If you want to add new extensions/services themselves to the Neutron Service, new YANG data models need to be added, but that is out of scope of this document because this guide is for a developer who will be *using* the feature to build something separate, but *not* somebody who will be developing code for this feature itself.

Neutron Service Architecture

The Neutron Service defines YANG models for OpenStack Neutron integration. When OpenStack admins/users request changes (creation/update/deletion) of Neutron resources, e.g., Neutron network, Neutron subnet, Neutron port, the corresponding YANG model within OpenDaylight will be modified. The OpenDaylight OpenStack will subscribe the changes on those models and will be notified those modification through MD-SAL when changes are made. Then the provider will do the necessary tasks to realize OpenStack integration. How to realize it (or even not realize it) is up to each provider. The Neutron Service itself does not take care of it.

How to Write a SB Neutron Consumer

In Boron, there is only one options for SB Neutron Consumers:

- Listening for changes via the Neutron YANG model

Until Beryllium there was another way with the legacy I*Aware interface. From Boron, the interface was eliminated. So all the SB Neutron Consumers have to use Neutron YANG model.

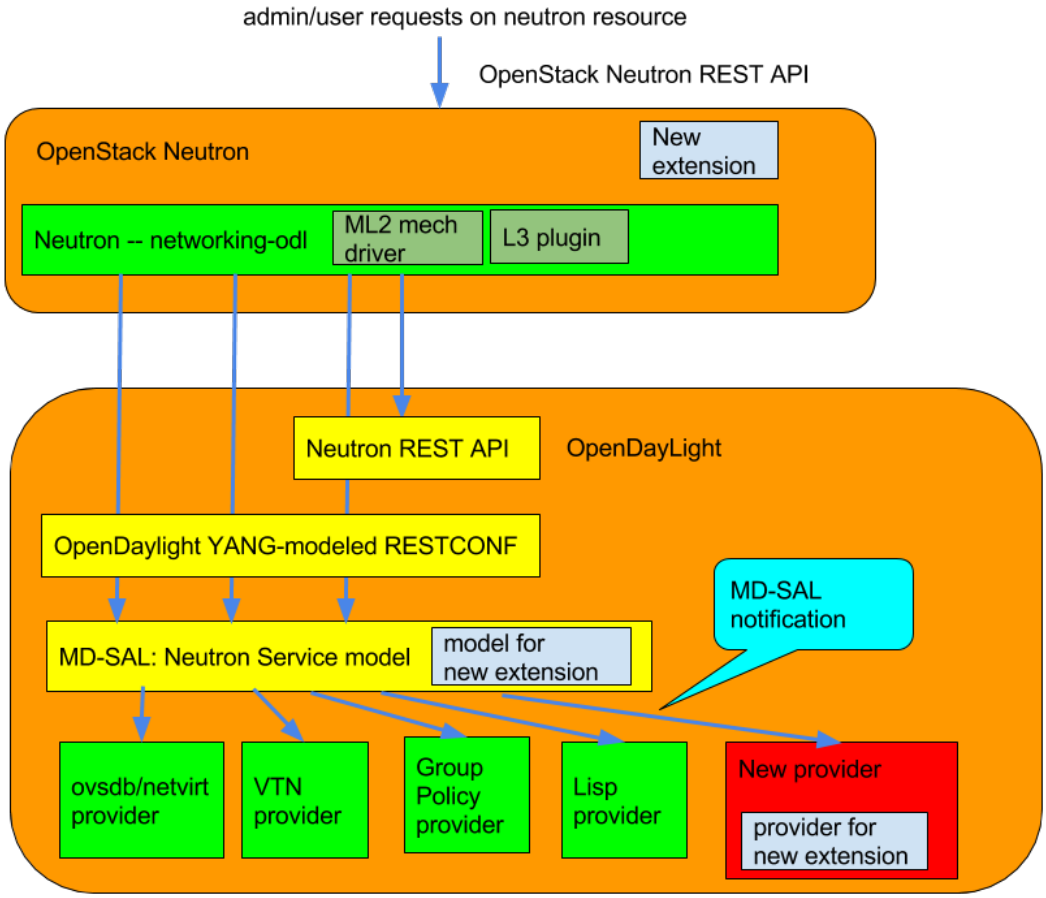

Fig. 1: Neutron Service Architecture

Neutron YANG models

Neutron service defines YANG models for Neutron. The details can be found at

- https://git.opendaylight.org/gerrit/gitweb?p=neutron.git;a=tree;f=model/src/main/yang;hb=refs/heads/stable/boron

Basically those models are based on OpenStack Neutron API definitions. For exact definitions, OpenStack Neutron source code needs to be referred as the above documentation does not always cover the necessary details. There is nothing special to utilize those Neutron YANG models. The basic procedure will be:

1. subscribe for changes made to the model
2. respond on the data change notification for each models

Note: Currently there is no way to refuse the request configuration at this point. That is left to future work.

```
public class NeutronNetworkChangeListener implements DataChangeListener, AutoCloseable {
    private ListenerRegistration<DataChangeListener> registration;
    private DataBroker db;

    public NeutronNetworkChangeListener(DataBroker db){
        this.db = db;
        // create identity path to register on service startup
        InstanceIdentifier<Network> path = InstanceIdentifier
                .create(Neutron.class)
                .child(Networks.class)
                .child(Network.class);
        LOG.debug("Register listener for Neutron Network model data changes");
        // register for Data Change Notification
        registration =
                this.db.registerDataChangeListener(LogicalDatastoreType.CONFIGURATION,␣
↪path, this, DataChangeScope.ONE);

    }

    @Override
    public void onDataChanged(
            AsyncDataChangeEvent<InstanceIdentifier<?>, DataObject> changes) {
        LOG.trace("Data changes : {}",changes);

        // handle data change notification
        Object[] subscribers = NeutronIAwareUtil.getInstances(INeutronNetworkAware.class,
↪ this);
        createNetwork(changes, subscribers);
        updateNetwork(changes, subscribers);
        deleteNetwork(changes, subscribers);
    }
}
```

Neutron configuration

From Boron, new models of configuration for OpenDaylight to tell OpenStack Neutron/`networking-odl` its configuration/capability.

`hostconfig`

This is for OpenDaylight to tell per-node configuration to Neutron. Especially this is used by pseudo agent port binding heavily.

The model definition can be found at

- https://git.opendaylight.org/gerrit/gitweb?p=neutron.git;a=blob;f=model/src/main/yang/neutron-hostconfig.yang;hb=refs/heads/stable/boron

How to populate this for pseudo agent port binding is documented at

- https://opendev.org/openstack/networking-odl/src/commit/7fd1258f6e161c035da41c8e95361648a0fb0d7c/doc/source/devref/hostconfig.rst

Neutron extension config

In Boron this is experimental. The model definition can be found at

- https://git.opendaylight.org/gerrit/gitweb?p=neutron.git;a=blob;f=model/src/main/yang/neutron-extensions.yang;hb=refs/heads/stable/boron

Each Neutron Service provider has its own feature set. Some support the full features of OpenStack, but others support only a subset. With same supported Neutron API, some functionality may or may not be supported. So there is a need for a way that OpenDaylight can tell `networking-odl` its capability. Thus `networking-odl` can initialize Neutron properly based on reported capability.

Neutron Logger

There is another small Karaf feature, `odl-neutron-logger`, which logs changes of Neutron YANG models. which can be used for debug/audit.

It would also help to understand how to listen the change.

- https://git.opendaylight.org/gerrit/gitweb?p=neutron.git;a=blob;f=neutron-logger/src/main/java/org/opendaylight/neutron/logger/NeutronLogger.java;hb=refs/heads/stable/boron

API Reference Documentation

The OpenStack Neutron API references

- https://docs.openstack.org/api-ref/network/v2/index.html
- https://docs.openstack.org/api-ref/network/v2/index.html#extensions

Neutron Northbound

How to add new API support

OpenStack Neutron is a moving target. It is continuously adding new features as new rest APIs. Here is a basic step to add new API support:

In the Neutron Northbound project:

- Add new YANG model for it under `neutron/model/src/main/yang` and `update neutron.yang`
- Add northbound API for it, and `neutron-spi`
  - Implement `Neutron<New API>Request.java` and `Neutron<New API>Norhtbound.java` under `neutron/northbound-api/src/main/java/org/opendaylight/neutron/northbound/api/`
  - Implement `INeutron<New API>CRUD.java` and new data structure if any under `neutron/neutron-spi/src/main/java/org/opendaylight/neutron/spi/`
  - update `neutron/neutron-spi/src/main/java/org/opendaylight/neutron/spi/NeutronCRUDInterfaces.java` to wire new CRUD interface
  - Add unit tests, `Neutron<New structure>JAXBTest.java` under `neutron/neutron-spi/src/test/java/org/opendaylight/neutron/spi/`
- update `neutron/northbound-api/src/main/java/org/opendaylight/neutron/northbound/api/NeutronNorthboundRSApplication.java` to wire new northbound API to `RSApplication`
- Add transcriber, `Neutron<New API>Interface.java` under `transcriber/src/main/java/org/opendaylight/neutron/transcriber/`
- update `transcriber/src/main/java/org/opendaylight/neutron/transcriber/NeutronTranscriberProvider.java` to wire a new transcriber
  - Add integration tests `Neutron<New API>Tests.java` under `integration/test/src/test/java/org/opendaylight/neutron/e2etest/`
  - update `integration/test/src/test/java/org/opendaylight/neutron/e2etest/ITNeutronE2E.java` to run a newly added tests.

In OpenStack `networking-odl`

- Add new driver (or plugin) for new API with tests.

In a southbound Neutron Provider

- implement actual back-end to realize those new API by listening related YANG models.

How to write transcriber

For each Neutron data object, there is an `Neutron*Interface` defined within the transcriber artifact that will write that object to the MD-SAL configuration datastore.

All `Neutron*Interface` extend `AbstractNeutronInterface`, in which two methods are defined:

- one takes the Neutron object as input, and will create a data object from it.
- one takes an uuid as input, and will create a data object containing the uuid.

```
protected abstract T toMd(S neutronObject);
protected abstract T toMd(String uuid);
```

In addition the `AbstractNeutronInterface` class provides several other helper methods (`addMd`, `updateMd`, `removeMd`), which handle the actual writing to the configuration datastore.

The semantics of the `toMD()` methods

Each of the Neutron YANG models defines structures containing data. Further each YANG-modeled structure has it own builder. A particular `toMD()` method instantiates an instance of the correct builder, fills in the properties of the builder from the corresponding values of the Neutron object and then creates the YANG-modeled structures via the `build()` method.

As an example, one of the `toMD` code for Neutron Networks is presented below:

```
protected Network toMd(NeutronNetwork network) {
    NetworkBuilder networkBuilder = new NetworkBuilder();
    networkBuilder.setAdminStateUp(network.getAdminStateUp());
    if (network.getNetworkName() != null) {
        networkBuilder.setName(network.getNetworkName());
    }
    if (network.getShared() != null) {
        networkBuilder.setShared(network.getShared());
    }
    if (network.getStatus() != null) {
        networkBuilder.setStatus(network.getStatus());
    }
    if (network.getSubnets() != null) {
        List<Uuid> subnets = new ArrayList<Uuid>();
        for( String subnet : network.getSubnets()) {
            subnets.add(toUuid(subnet));
        }
        networkBuilder.setSubnets(subnets);
    }
    if (network.getTenantID() != null) {
        networkBuilder.setTenantId(toUuid(network.getTenantID()));
    }
    if (network.getNetworkUUID() != null) {
        networkBuilder.setUuid(toUuid(network.getNetworkUUID()));
    } else {
        logger.warn("Attempting to write neutron network without UUID");
    }
    return networkBuilder.build();
}
```

ODL Parent Developer Guide

Parent POMs

Overview

The ODL Parent component for OpenDaylight provides a number of Maven parent POMs which allow Maven projects to be easily integrated in the OpenDaylight ecosystem. Technically, the aim of projects in OpenDaylight is to produce Karaf features, and these parent projects provide common support for the different types of projects involved.

These parent projects are:

- `odlparent-lite` — the basic parent POM for Maven modules which don't produce artifacts (*e.g.* aggregator POMs)
- `odlparent` — the common parent POM for Maven modules containing Java code
- `bundle-parent` — the parent POM for Maven modules producing OSGi bundles

The following parent projects are deprecated, but still used in Carbon:

- `feature-parent` — the parent POM for Maven modules producing Karaf 3 feature repositories
- `karaf-parent` — the parent POM for Maven modules producing Karaf 3 distributions

The following parent projects are new in Carbon, for Karaf 4 support (which won't be complete until Nitrogen):

- `single-feature-parent` — the parent POM for Maven modules producing a single Karaf 4 feature
- `feature-repo-parent` — the parent POM for Maven modules producing Karaf 4 feature repositories
- `karaf4-parent` — the parent POM for Maven modules producing Karaf 4 distributions

odlparent-lite

This is the base parent for all OpenDaylight Maven projects and modules. It provides the following, notably to allow publishing artifacts to Maven Central:

- license information;
- organization information;
- issue management information (a link to our Bugzilla);
- continuous integration information (a link to our Jenkins setup);
- default Maven plugins (`maven-clean-plugin`, `maven-deploy-plugin`, `maven-install-plugin`, `maven-javadoc-plugin` with HelpMojo support, `maven-project-info-reports-plugin`, `maven-site-plugin` with Asciidoc support, `jdepend-maven-plugin`);
- distribution management information.

It also defines two profiles which help during development:

- `q` (`-Pq`), the quick profile, which disables tests, code coverage, Javadoc generation, code analysis, etc. — anything which is not necessary to build the bundles and features (see this blog post for details);
- `addInstallRepositoryPath` (`-DaddInstallRepositoryPath=.../karaf/system`) which can be used to drop a bundle in the appropriate Karaf location, to enable hot-reloading of bundles during development (see this blog post for details).

For modules which don't produce any useful artifacts (*e.g.* aggregator POMs), you should add the following to avoid processing artifacts:

```
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-deploy-plugin</artifactId>
            <configuration>
                <skip>true</skip>
            </configuration>
```

(continues on next page)

(continued from previous page)

```
        </plugin>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-install-plugin</artifactId>
            <configuration>
                <skip>true</skip>
            </configuration>
        </plugin>
    </plugins>
</build>
```

odlparent

This inherits from `odlparent-lite` and mainly provides dependency and plugin management for OpenDaylight projects.

If you use any of the following libraries, you should rely on `odlparent` to provide the appropriate versions:

- Akka (and Scala)
- Apache Commons:
  - `commons-codec`
  - `commons-fileupload`
  - `commons-io`
  - `commons-lang`
  - `commons-lang3`
  - `commons-net`
- Apache Shiro
- Guava
- JAX-RS with Jersey
- JSON processing:
  - GSON
  - Jackson
- Logging:
  - Logback
  - SLF4J
- Netty
- OSGi:
  - Apache Felix
  - core OSGi dependencies (`core`, `compendium`...)
- Testing:
  - Hamcrest

  - JSON assert
  - JUnit
  - Mockito
  - Pax Exam
  - PowerMock
- XML/XSL:
  - Xerces
  - XML APIs

Note: This list is not exhaustive. It is also not cast in stone;if you would like to add a new dependency (or migrate a dependency), please contact the mailing list.

`odlparent` also enforces some Checkstyle verification rules. In particular, it enforces the common license header used in all OpenDaylight code:

```
/*
 * Copyright © ${year} ${holder} and others.  All rights reserved.
 *
 * This program and the accompanying materials are made available under the
 * terms of the Eclipse Public License v1.0 which accompanies this distribution,
 * and is available at http://www.eclipse.org/legal/epl-v10.html
 */
```

where "`${year}`" is initially the first year of publication, then (after a year has passed) the first and latest years of publication, separated by commas (*e.g.* "2014, 2016"), and "`${holder}`" is the initial copyright holder (typically, the first author's employer). "All rights reserved" is optional.

If you need to disable this license check, *e.g.* for files imported under another license (EPL-compatible of course), you can override the `maven-checkstyle-plugin` configuration. `features-test` does this for its `CustomBundleUrlStreamHandlerFactory` class, which is ASL-licensed:

```
<plugin>
    <artifactId>maven-checkstyle-plugin</artifactId>
    <executions>
        <execution>
            <id>check-license</id>
            <goals>
                <goal>check</goal>
            </goals>
            <phase>process-sources</phase>
            <configuration>
                <configLocation>check-license.xml</configLocation>
                <headerLocation>EPL-LICENSE.regexp.txt</headerLocation>
                <includeResources>false</includeResources>
                <includeTestResources>false</includeTestResources>
                <sourceDirectory>${project.build.sourceDirectory}</sourceDirectory>
                <excludes>
                    <!-- Skip Apache Licensed files -->
                    org/opendaylight/odlparent/featuretest/
↪CustomBundleUrlStreamHandlerFactory.java
```

(continues on next page)

(continued from previous page)

```
                </excludes>
                <failsOnError>false</failsOnError>
                <consoleOutput>true</consoleOutput>
            </configuration>
        </execution>
    </executions>
</plugin>
```

bundle-parent

This inherits from `odlparent` and enables functionality useful for OSGi bundles:

- `maven-javadoc-plugin` is activated, to build the Javadoc JAR;
- `maven-source-plugin` is activated, to build the source JAR;
- `maven-bundle-plugin` is activated (including extensions), to build OSGi bundles (using the "bundle" packaging).

In addition to this, JUnit is included as a default dependency in "test" scope.

features-parent

This inherits from `odlparent` and enables functionality useful for Karaf features:

- `karaf-maven-plugin` is activated, to build Karaf features — but for OpenDaylight, projects need to use **"jar"** packaging (**not** "feature" or "kar");
- `features.xml` files are processed from templates stored in `src/main/features/features.xml`;
- Karaf features are tested after build to ensure they can be activated in a Karaf container.

The `features.xml` processing allows versions to be omitted from certain feature dependencies, and replaced with "{{version}}". For example:

```
<features name="odl-mdsal-${project.version}" xmlns="http://karaf.apache.org/xmlns/
↪features/v1.2.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://karaf.apache.org/xmlns/features/v1.2.0 http://karaf.apache.
↪org/xmlns/features/v1.2.0">

    <repository>mvn:org.opendaylight.odlparent/features-odlparent/{{VERSION}}/xml/
↪features</repository>

    [...]
    <feature name='odl-mdsal-broker-local' version='${project.version}' description=
↪"OpenDaylight :: MDSAL :: Broker">
        <feature version='${yangtools.version}'>odl-yangtools-common</feature>
        <feature version='${mdsal.version}'>odl-mdsal-binding-dom-adapter</feature>
        <feature version='${mdsal.model.version}'>odl-mdsal-models</feature>
        <feature version='${project.version}'>odl-mdsal-common</feature>
        <feature version='${config.version}'>odl-config-startup</feature>
        <feature version='${config.version}'>odl-config-netty</feature>
        <feature version='[3.3.0,4.0.0)'>odl-lmax</feature>
```

(continues on next page)

(continued from previous page)

```
        [...]
        <bundle>mvn:org.opendaylight.controller/sal-dom-broker-config/{{VERSION}}</
↪bundle>
        <bundle start-level="40">mvn:org.opendaylight.controller/blueprint/{{VERSION}}</
↪bundle>
        <configfile finalname="${config.configfile.directory}/${config.mdsal.configfile}
↪">mvn:org.opendaylight.controller/md-sal-config/{{VERSION}}/xml/config</configfile>
    </feature>
```

As illustrated, versions can be omitted in this way for repository dependencies, bundle dependencies and configuration files. They must be specified traditionally (either hard-coded, or using Maven properties) for feature dependencies.

karaf-parent

This allows building a Karaf 3 distribution, typically for local testing purposes. Any runtime-scoped feature dependencies will be included in the distribution, and the `karaf.localFeature` property can be used to specify the boot feature (in addition to `standard`).

single-feature-parent

This inherits from `odlparent` and enables functionality useful for Karaf 4 features:

- `karaf-maven-plugin` is activated, to build Karaf features, typically with `"feature"` packaging (`"kar"` is also supported);
- `feature.xml` files are generated based on the compile-scope dependencies defined in the POM, optionally initialized from a stub in `src/main/feature/feature.xml`.
- Karaf features are tested after build to ensure they can be activated in a Karaf container.

The `feature.xml` processing adds transitive dependencies by default, which allows features to be defined using only the most significant dependencies (those that define the feature); other requirements are determined automatically as long as they exist as Maven dependencies.

`configfiles` need to be defined both as Maven dependencies (with the appropriate type and classifier) and as `<configfile>` elements in the `feature.xml` stub.

Other features which a feature depends on need to be defined as Maven dependencies with type "xml" and classifier "features" (note the plural here).

feature-repo-parent

This inherits from `odlparent` and enables functionality useful for Karaf 4 feature repositories. It follows the same principles as `single-feature-parent`, but is designed specifically for repositories and should be used only for this type of artifacts.

It builds a feature repository referencing all the (feature) dependencies listed in the POM.

karaf4-parent

This allows building a Karaf 4 distribution, typically for local testing purposes. Any runtime-scoped feature dependencies will be included in the distribution, and the `karaf.localFeature` property can be used to specify the boot feature (in addition to `standard`).

Features (for Karaf 3)

The ODL Parent component for OpenDaylight provides a number of Karaf 3 features which can be used by other Karaf 3 features to use certain third-party upstream dependencies.

These features are:

- Akka features (in the `features-akka` repository):
  - `odl-akka-all` — all Akka bundles;
  - `odl-akka-scala-2.11` — Scala runtime for OpenDaylight;
  - `odl-akka-system-2.4` — Akka actor framework bundles;
  - `odl-akka-clustering-2.4` — Akka clustering bundles and dependencies;
  - `odl-akka-leveldb-0.7` — LevelDB;
  - `odl-akka-persistence-2.4` — Akka persistence;
- general third-party features (in the `features-odlparent` repository):
  - `odl-netty-4` — all Netty bundles;
  - `odl-guava-18` — Guava 18;
  - `odl-guava-21` — Guava 21 (not intended for use in Carbon);
  - `odl-lmax-3` — LMAX Disruptor;
  - `odl-triemap-0.2` — Concurrent Hash-Trie Map.

To use these, you need to declare a dependency on the appropriate repository in your `features.xml` file:

```
<repository>mvn:org.opendaylight.odlparent/features-odlparent/{{VERSION}}/xml/features</
↪repository>
```

and then include the feature, *e.g.*:

```
<feature name='odl-mdsal-broker-local' version='${project.version}' description=
↪"OpenDaylight :: MDSAL :: Broker">
    [...]
    <feature version='[3.3.0,4.0.0)'>odl-lmax</feature>
    [...]
</feature>
```

You also need to depend on the features repository in your POM:

```
<dependency>
    <groupId>org.opendaylight.odlparent</groupId>
    <artifactId>features-odlparent</artifactId>
    <classifier>features</classifier>
```

(continues on next page)

(continued from previous page)

```
    <type>xml</type>
</dependency>
```

assuming the appropriate dependency management:

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.opendaylight.odlparent</groupId>
            <artifactId>odlparent-artifacts</artifactId>
            <version>1.8.0-SNAPSHOT</version>
            <scope>import</scope>
            <type>pom</type>
        </dependency>
    </dependencies>
</dependencyManagement>
```

(the version number there is appropriate for Carbon). For the time being you also need to depend separately on the individual JARs as compile-time dependencies to build your dependent code; the relevant dependencies are managed in `odlparent`'s dependency management.

The suggested version ranges are as follows:

- `odl-netty`: `[4.0.37,4.1.0)` or `[4.0.37,5.0.0)`;
- `odl-guava`: `[18,19)` (if your code is ready for it, `[19,20)` is also available, but the current default version of Guava in OpenDaylight is 18);
- `odl-lmax`: `[3.3.4,4.0.0)`

Features (for Karaf 4)

There are equivalent features to all the Karaf 3 features, for Karaf 4. The repositories use "features4" instead of "features", and the features use `odl4` instead of `odl`.

The following new features are specific to Karaf 4:

- Karaf wrapper features (also in the `features4-odlparent` repository) — these can be used to pull in a Karaf feature using a Maven dependency in a POM:
  - `odl-karaf-feat-feature` — the Karaf `feature` feature;
  - `odl-karaf-feat-jdbc` — the Karaf `jdbc` feature;
  - `odl-karaf-feat-jetty` — the Karaf `jetty` feature;
  - `odl-karaf-feat-war` — the Karaf `war` feature.

To use these, all you need to do now is add the appropriate dependency in your feature POM; for example:

```
<dependency>
    <groupId>org.opendaylight.odlparent</groupId>
    <artifactId>odl4-guava-18</artifactId>
    <classifier>features</classifier>
```

(continues on next page)

(continued from previous page)

```
    <type>xml</type>
</dependency>
```

assuming the appropriate dependency management:

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.opendaylight.odlparent</groupId>
            <artifactId>odlparent-artifacts</artifactId>
            <version>1.8.0-SNAPSHOT</version>
            <scope>import</scope>
            <type>pom</type>
        </dependency>
    </dependencies>
</dependencyManagement>
```

(the version number there is appropriate for Carbon). We no longer use version ranges, the feature dependencies all use the `odlparent` version (but you should rely on the artifacts POM).

YANG Tools Developer Guide

Overview

YANG Tools is set of libraries and tooling providing support for use YANG for Java (or other JVM-based language) projects and applications. The YANG Tools provides following features in OpenDaylight:

- Parsing of YANG sources and semantic inference of relationship across YANG models as defined in RFC6020
- Representation of YANG-modeled data in Java
  - **Normalized Node** representation - DOM-like tree model, which uses conceptual meta-model more tailored to YANG and OpenDaylight use-cases than a standard XML DOM model allows for.
- Serialization / deserialization of YANG-modeled data driven by YANG models
  - XML - as defined in RFC6020
  - JSON - as defined in draft-lhotka-netmod-yang-json-01
  - Support for third-party generators processing YANG models.

Architecture

YANG tools consist of the following logical subsystems:

Table 45: **YANG Tools**

Tool	Description
Commons	Set of general purpose code, which is not specific to YANG, but is also useful outside YANG Tools implementation.
YANG Model and Parser	YANG semantic model and lexical and semantic parser of YANG models, which creates in-memory cross-referenced representation of YANG models, which is used by other components to determine their behavior based on the model.
YANG data	Definition of Normalized Node APIs and Data Tree APIs, reference implementation of these APIs and implementation of XML and JSON codecs for Normalized Nodes.
YANG Maven Plugin	Maven plugin which integrates YANG parser into Maven build life-cycle and provides code-generation framework for components, which wants to generate code or other artefacts based on YANG model.

Concepts

Project defines base concepts and helper classes which are project-agnostic and could be used outside of YANG Tools project scope.

Components

- `yang-common`
- `yang-data-api`
- `yang-data-codec-gson`
- `yang-data-codec-xml`
- `yang-data-impl`
- `yang-data-jaxen`
- `yang-data-transform`
- `yang-data-util`
- `yang-maven-plugin`
- `yang-maven-plugin-it`
- `yang-maven-plugin-spi`
- `yang-model-api`
- `yang-model-export`
- `yang-model-util`
- `yang-parser-api`
- `yang-parser-impl`

YANG Model API

Class diagram of yang model API

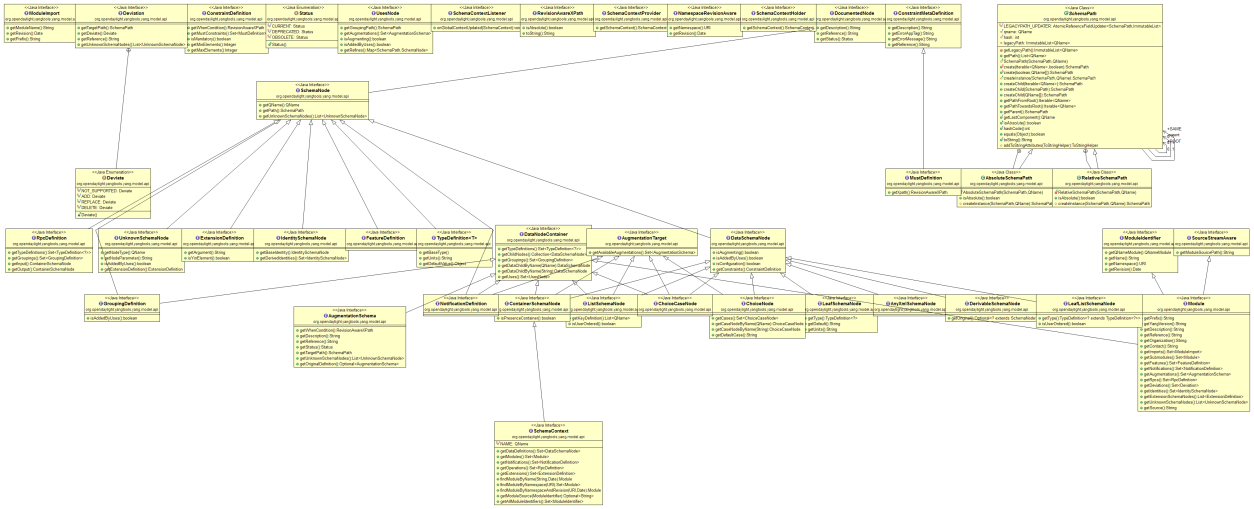

Fig. 2: YANG Model API

YANG Parser

YANG Statement Parser works on the idea of statement concepts as defined in RFC6020, section 6.3. We come up here with basic ModelStatement and StatementDefinition, following RFC6020 idea of having sequence of statements, where every statement contains keyword and zero or one argument. ModelStatement is extended by DeclaredStatement (as it comes from source, e.g. YANG source) and EffectiveStatement, which contains other sub-statements and tends to represent result of semantic processing of other statements (uses, augment for YANG). IdentifierNamespace represents common superclass for YANG model namespaces.

Input of the YANG Statement Parser is a collection of StatementStreamSource objects. StatementStreamSource interface is used for inference of effective model and is required to emit its statements using supplied StatementWriter. Each source (e.g. YANG source) has to be processed in three steps in order to emit different statements for each step. This package provides support for various namespaces used across statement parser in order to map relations during declaration phase process.

Currently, there are two implementations of StatementStreamSource in YANGtools:

- YangStatementSourceImpl - intended for yang sources
- YinStatementSourceImpl - intended for yin sources

YANG data API

Class diagram of yang data API

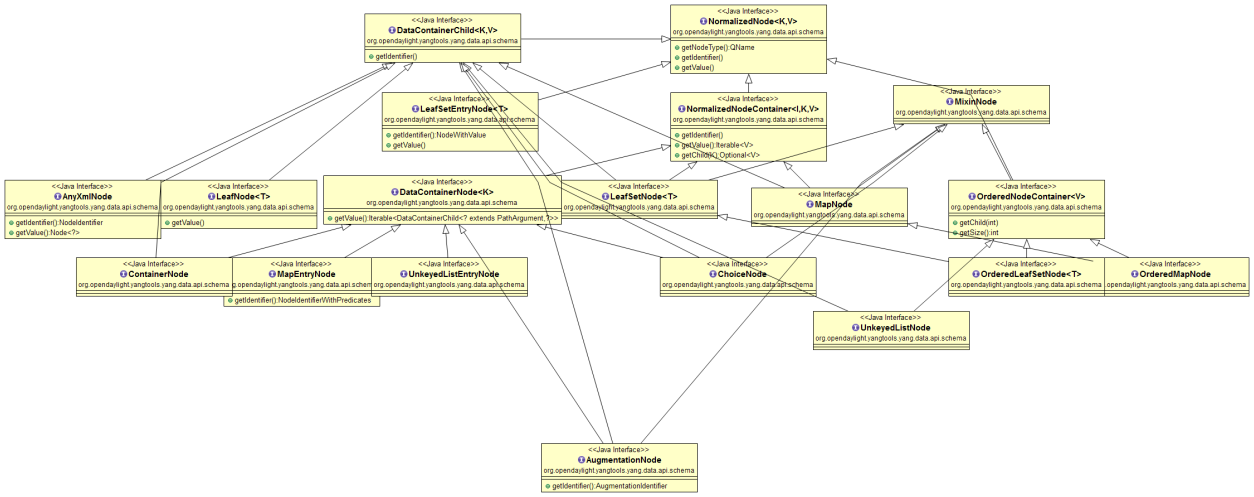

Fig. 3: YANG data API

YANG data Codecs

Codecs which enable serialization of NormalizedNodes into YANG-modeled data in XML or JSON format and deserialization of YANG-modeled data in XML or JSON format into NormalizedNodes.

YANG Maven Plugin

Maven plugin which integrates YANG parser into Maven build life-cycle and provides code-generation framework for components, which wants to generate code or other artefacts based on YANG model.

How to / Tutorials

Working with YANG Model

First thing you need to do if you want to work with YANG models is to instantiate a SchemaContext object. This object type describes one or more parsed YANG modules.

In order to create it you need to utilize YANG statement parser which takes one or more StatementStreamSource objects as input and then produces the SchemaContext object.

StatementStreamSource object contains the source file information. It has two implementations, one for YANG sources - YangStatementSourceImpl, and one for YIN sources - YinStatementSourceImpl.

Here is an example of creating StatementStreamSource objects for YANG files, providing them to the YANG statement parser and building the SchemaContext:

```
StatementStreamSource yangModuleSource == new YangStatementSourceImpl("/example.yang",␣
↪false);
StatementStreamSource yangModuleSource2 == new YangStatementSourceImpl("/example2.yang",␣
↪false);

CrossSourceStatementReactor.BuildAction reactor == YangInferencePipeline.RFC6020_REACTOR.
↪newBuild();
```

(continues on next page)

(continued from previous page)

```
reactor.addSources(yangModuleSource, yangModuleSource2);

SchemaContext schemaContext == reactor.buildEffective();
```

First, StatementStreamSource objects with two constructor arguments should be instantiated: path to the yang source file (which is a regular String object) and a Boolean which determines if the path is absolute or relative.

Next comes the initiation of new yang parsing cycle - which is represented by CrossSourceStatementReactor.BuildAction object. You can get it by calling method `newBuild()` on CrossSourceStatementReactor object (RFC6020_REACTOR) in YangInferencePipeline class.

Then you should feed yang sources to it by calling method `addSources()` that takes one or more StatementStreamSource objects as arguments.

Finally, you call the method `buildEffective()` on the reactor object which returns `EffectiveSchemaContext` (that is a concrete implementation of `SchemaContext`). Now you are ready to work with contents of the added YANG sources.

Let us explain how to work with models contained in the newly created `SchemaContext`. If you want to get all the modules in the `schemaContext`, you have to call method `getModules()` which returns a Set of modules. If you want to get all the data definitions in `schemaContext`, you need to call method `getDataDefinitions`, etc.

```
Set<Module> modules == schemaContext.getModules();
Set<DataSchemaNodes> dataSchemaNodes == schemaContext.getDataDefinitions();
```

Usually you want to access specific modules. Getting a concrete module from SchemaContext is a matter of calling one of these methods:

- `findModuleByName()`,
- `findModuleByNamespace()`,
- `findModuleByNamespaceAndRevision()`.

In the first case, you need to provide module name as it is defined in the yang source file and module revision date if it specified in the yang source file (if it is not defined, you can just pass a null value). In order to provide the revision date in proper format, you can use a utility class named SimpleDateFormatUtil.

```
Module exampleModule == schemaContext.findModuleByName("example-module", null);
// or
Date revisionDate == SimpleDateFormatUtil.getRevisionFormat().parse("2015-09-02");
Module exampleModule == schemaContext.findModuleByName("example-module", revisionDate);
```

In the second case, you have to provide module namespace in form of an URI object.

```
Module exampleModule == schema.findModuleByNamespace(new URI("opendaylight.org/example-
↪module"));
```

In the third case, you provide both module namespace and revision date as arguments.

Once you have a Module object, you can access its contents as they are defined in YANG Model API. One way to do this is to use method like `getIdentities()` or `getRpcs()` which will give you a Set of objects. Otherwise you can access a DataSchemaNode directly via the method `getDataChildByName()` which takes a `QName` object as its only argument. Here are a few examples.

```
Set<AugmentationSchema> augmentationSchemas == exampleModule.getAugmentations();
Set<ModuleImport> moduleImports == exampleModule.getImports();
```

(continues on next page)

(continued from previous page)

```
ChoiceSchemaNode choiceSchemaNode == (ChoiceSchemaNode) exampleModule.
↪getDataChildByName(QName.create(exampleModule.getQNameModule(), "example-choice"));

ContainerSchemaNode containerSchemaNode == (ContainerSchemaNode) exampleModule.
↪getDataChildByName(QName.create(exampleModule.getQNameModule(), "example-container"));
```

The YANG statement parser can work in three modes:

- default mode
- mode with active resolution of if-feature statements
- mode with active semantic version processing

The default mode is active when you initialize the parsing cycle as usual by calling the method `newBuild()` without passing any arguments to it. The second and third mode can be activated by invoking the `newBuild()` with a special argument. You can either activate just one of them or both by passing proper arguments. Let us explain how these modes work.

Mode with active resolution of if-features makes yang statements containing an if-feature statement conditional based on the supported features. These features are provided in the form of a `QName-based java.util.Set` object. In the example below, only two features are supported: example-feature-1 and example-feature-2. The Set which contains this information is passed to the method `newBuild()` and the mode is activated.

```
Set<QName> supportedFeatures = ImmutableSet.of(
    QName.create("example-namespace", "2016-08-31", "example-feature-1"),
    QName.create("example-namespace", "2016-08-31", "example-feature-2"));

CrossSourceStatementReactor.BuildAction reactor = YangInferencePipeline.RFC6020_REACTOR.
↪newBuild(supportedFeatures);
```

In case when no features should be supported, you should provide an empty `Set<QName>` object.

```
Set<QName> supportedFeatures = ImmutableSet.of();

CrossSourceStatementReactor.BuildAction reactor = YangInferencePipeline.RFC6020_REACTOR.
↪newBuild(supportedFeatures);
```

When this mode is not activated, all features in the processed YANG sources are supported.

Mode with active semantic version processing changes the way how YANG import statements work - each module import is processed based on the specified semantic version statement and the revision-date statement is ignored. In order to activate this mode, you have to provide StatementParserMode.SEMVER_MODE enum constant as argument to the method `newBuild()`.

```
CrossSourceStatementReactor.BuildAction reactor == YangInferencePipeline.RFC6020_REACTOR.
↪newBuild(StatementParserMode.SEMVER_MODE);
```

Before you use a semantic version statement in a YANG module, you need to define an extension for it so that the YANG statement parser can recognize it.

```
module semantic-version {
    namespace "urn:opendaylight:yang:extension:semantic-version";
    prefix sv;
    yang-version 1;
```

(continues on next page)

(continued from previous page)

```
    revision 2016-02-02 {
        description "Initial version";
    }
    sv:semantic-version "0.0.1";

    extension semantic-version {
        argument "semantic-version" {
            yin-element false;
        }
    }
}
```

In the example above, you see a YANG module which defines semantic version as an extension. This extension can be imported to other modules in which we want to utilize the semantic versioning concept.

Below is a simple example of the semantic versioning usage. With semantic version processing mode being active, the foo module imports the bar module based on its semantic version. Notice how both modules import the module with the semantic-version extension.

```
module foo {
    namespace foo;
    prefix foo;
    yang-version 1;

    import semantic-version { prefix sv; revision-date 2016-02-02; sv:semantic-version
↪"0.0.1"; }
    import bar { prefix bar; sv:semantic-version "0.1.2";}

    revision "2016-02-01" {
        description "Initial version";
    }
    sv:semantic-version "0.1.1";

    ...
}
```

```
module bar {
    namespace bar;
    prefix bar;
    yang-version 1;

    import semantic-version { prefix sv; revision-date 2016-02-02; sv:semantic-version
↪"0.0.1"; }

    revision "2016-01-01" {
        description "Initial version";
    }
    sv:semantic-version "0.1.2";

    ...
}
```

Every semantic version must have the following form: x.y.z. The x corresponds to a major version, the y corresponds to a minor version and the z corresponds to a patch version. If no semantic version is specified in a module or an import statement, then the default one is used - 0.0.0.

A major version number of 0 indicates that the model is still in development and is subject to change.

Following a release of major version 1, all modules will increment major version number when backwards incompatible changes to the model are made.

The minor version is changed when features are added to the model that do not impact current clients use of the model.

The patch version is incremented when non-feature changes (such as bugfixes or clarifications of human-readable descriptions that do not impact model functionality) are made that maintain backwards compatibility.

When importing a module with activated semantic version processing mode, only the module with the newest (highest) compatible semantic version is imported. Two semantic versions are compatible when all of the following conditions are met:

- the major version in the import statement and major version in the imported module are equal. For instance, 1.5.3 is compatible with 1.5.3, 1.5.4, 1.7.2, etc., but it is not compatible with 0.5.2 or 2.4.8, etc.
- the combination of minor version and patch version in the import statement is not higher than the one in the imported module. For instance, 1.5.2 is compatible with 1.5.2, 1.5.4, 1.6.8 etc. In fact, 1.5.2 is also compatible with versions like 1.5.1, 1.4.9 or 1.3.7 as they have equal major version. However, they will not be imported because their minor and patch version are lower (older).

If the import statement does not specify a semantic version, then the default one is chosen - 0.0.0. Thus, the module is imported only if it has a semantic version compatible with the default one, for example 0.0.0, 0.1.3, 0.3.5 and so on.

Working with YANG data

If you want to work with YANG data, you are going to need NormalizedNode objects that are specified in the YANG data API. NormalizedNode is an interface at the top of the YANG data hierarchy. It is extended through sub-interfaces which define the behavior of specific NormalizedNode types like AnyXmlNode, ChoiceNode, LeafNode, ContainerNode, etc. Concrete implementations of these interfaces are defined in `yang-data-impl` module. Once you have one or more NormalizedNode instances, you can perform CRUD operations on YANG data tree which is an in-memory database designed to store normalized nodes in a tree-like structure.

In some cases it, is clear which NormalizedNode type belongs to which yang statement (e.g. AnyXmlNode, ChoiceNode, LeafNode). However, there are some normalized nodes which are named differently from their yang counterparts. They are listed below:

Table 46: **Normalized Nodes**

Node	Description
LeafSetNode	Leaf-list
OrderedLeafSetNode	Leaf-list that is ordered-by user
LeafSetEntryNode	Concrete entry in a leaf-list
MapNode	Keyed list
OrderedMapNode	Keyed list that is ordered-by user
MapEntryNode	Concrete entry in a keyed list
UnkeyedListNode	Unkeyed list
UnkeyedListEntryNode	Concrete entry in an unkeyed list

To create a concrete NormalizedNode object, use the utility class Builders or ImmutableNodes. These classes can be found in *yang-data-impl* module and they provide methods for building each type of normalized node. Here is a simple example of building a normalized node:

```
// example 1
ContainerNode containerNode == Builders.containerBuilder().withNodeIdentifier(new␣
↪YangInstanceIdentifier.NodeIdentifier(QName.create(moduleQName, "example-container")).
↪build();

// example 2
ContainerNode containerNode2 == Builders.containerBuilder(containerSchemaNode).build();
```

Both examples produce the same result. NodeIdentifier is one of the four types of YangInstanceIdentifier (these types are described in the Javadoc of YangInstanceIdentifier). The purpose of YangInstanceIdentifier is to uniquely identify a particular node in the data tree. In the first example, you have to add NodeIdentifier before building the resulting node. In the second example it is also added using the provided ContainerSchemaNode object.

ImmutableNodes class offers similar builder methods and also adds an overloaded method called `fromInstanceId()` which allows you to create a NormalizedNode object based on YangInstanceIdentifier and SchemaContext. Below is an example which shows the use of this method.

```
YangInstanceIdentifier.NodeIdentifier contId == new YangInstanceIdentifier.
↪NodeIdentifier(QName.create(moduleQName, "example-container");

NormalizedNode<?, ?> contNode == ImmutableNodes.fromInstanceId(schemaContext,␣
↪YangInstanceIdentifier.create(contId));
```

Let us show a more complex example of creating a NormalizedNode. First, consider the following YANG module:

```
module example-module {
    namespace "opendaylight.org/example-module";
    prefix "example";

    container parent-container {
        container child-container {
            list parent-ordered-list {
                ordered-by user;

                key "parent-key-leaf";

                leaf parent-key-leaf {
                    type string;
                }

                leaf parent-ordinary-leaf {
                    type string;
                }

                list child-ordered-list {
                    ordered-by user;

                    key "child-key-leaf";

                    leaf child-key-leaf {
                        type string;
                    }

                    leaf child-ordinary-leaf {
```

(continues on next page)

(continued from previous page)

```
                        type string;
                    }
                }
            }
        }
    }
}
```

In the following example, two normalized nodes based on the module above are written to and read from the data tree.

```
TipProducingDataTree inMemoryDataTree ==     InMemoryDataTreeFactory.getInstance().
↪create(TreeType.OPERATIONAL);
inMemoryDataTree.setSchemaContext(schemaContext);

// first data tree modification
MapEntryNode parentOrderedListEntryNode == Builders.mapEntryBuilder().withNodeIdentifier(
new YangInstanceIdentifier.NodeIdentifierWithPredicates(
parentOrderedListQName, parentKeyLeafQName, "pkval1"))
.withChild(Builders.leafBuilder().withNodeIdentifier(
new YangInstanceIdentifier.NodeIdentifier(parentOrdinaryLeafQName))
.withValue("plfval1").build()).build();

OrderedMapNode parentOrderedListNode == Builders.orderedMapBuilder().withNodeIdentifier(
new YangInstanceIdentifier.NodeIdentifier(parentOrderedListQName))
.withChild(parentOrderedListEntryNode).build();

ContainerNode parentContainerNode == Builders.containerBuilder().withNodeIdentifier(
new YangInstanceIdentifier.NodeIdentifier(parentContainerQName))
.withChild(Builders.containerBuilder().withNodeIdentifier(
new NodeIdentifier(childContainerQName)).withChild(parentOrderedListNode).build()).
↪build();

YangInstanceIdentifier path1 == YangInstanceIdentifier.of(parentContainerQName);

DataTreeModification treeModification == inMemoryDataTree.takeSnapshot().
↪newModification();
treeModification.write(path1, parentContainerNode);

// second data tree modification
MapEntryNode childOrderedListEntryNode == Builders.mapEntryBuilder().withNodeIdentifier(
new YangInstanceIdentifier.NodeIdentifierWithPredicates(
childOrderedListQName, childKeyLeafQName, "chkval1"))
.withChild(Builders.leafBuilder().withNodeIdentifier(
new YangInstanceIdentifier.NodeIdentifier(childOrdinaryLeafQName))
.withValue("chlfval1").build()).build();

OrderedMapNode childOrderedListNode == Builders.orderedMapBuilder().withNodeIdentifier(
new YangInstanceIdentifier.NodeIdentifier(childOrderedListQName))
.withChild(childOrderedListEntryNode).build();

ImmutableMap.Builder<QName, Object> builder == ImmutableMap.builder();
ImmutableMap<QName, Object> keys == builder.put(parentKeyLeafQName, "pkval1").build();
```

(continues on next page)

(continued from previous page)

```
YangInstanceIdentifier path2 == YangInstanceIdentifier.of(parentContainerQName).
↪node(childContainerQName)
.node(parentOrderedListQName).node(new␣
↪NodeIdentifierWithPredicates(parentOrderedListQName, keys)).
↪node(childOrderedListQName);

treeModification.write(path2, childOrderedListNode);
treeModification.ready();
inMemoryDataTree.validate(treeModification);
inMemoryDataTree.commit(inMemoryDataTree.prepare(treeModification));

DataTreeSnapshot snapshotAfterCommits == inMemoryDataTree.takeSnapshot();
Optional<NormalizedNode<?, ?>> readNode == snapshotAfterCommits.readNode(path1);
Optional<NormalizedNode<?, ?>> readNode2 == snapshotAfterCommits.readNode(path2);
```

First comes the creation of in-memory data tree instance. The schema context (containing the model mentioned above) of this tree is set. After that, two normalized nodes are built. The first one consists of a parent container, a child container and a parent ordered list which contains a key leaf and an ordinary leaf. The second normalized node is a child ordered list that also contains a key leaf and an ordinary leaf.

In order to add a child node to a node, method `withChild()` is used. It takes a NormalizedNode as argument. When creating a list entry, YangInstanceIdentifier.NodeIdentifierWithPredicates should be used as its identifier. Its arguments are the `QName` of the list, `QName` of the list key and the value of the key. Method `withValue()` specifies a value for the ordinary leaf in the list.

Before writing a node to the data tree, a path (YangInstanceIdentifier) which determines its place in the data tree needs to be defined. The path of the first normalized node starts at the parent container. The path of the second normalized node points to the child ordered list contained in the parent ordered list entry specified by the key value `"pkval1"`.

Write operation is performed with both normalized nodes mentioned earlier. It consists of several steps. The first step is to instantiate a DataTreeModification object based on a DataTreeSnapshot. DataTreeSnapshot gives you the current state of the data tree. Then comes the write operation which writes a normalized node at the provided path in the data tree. After doing both write operations, method ready() has to be called, marking the modification as ready for application to the data tree. No further operations within the modification are allowed. The modification is then validated - checked whether it can be applied to the data tree. Finally, we commit it to the data tree.

Now you can access the written nodes. In order to do this, you must create a new DataTreeSnapshot instance and call the method `readNode()` with path argument pointing to a node in the tree.

Serialization / deserialization of YANG data

If you want to deserialize YANG-modeled data that has the form of an XML document, you can use the XML parser found in the module `yang-data-codec-xml`. The parser walks through the XML document containing YANG-modeled data based on the provided SchemaContext and emits node events into a NormalizedNodeStreamWriter. The parser disallows multiple instances of the same element except for leaf-list and list entries. The parser also expects that the YANG-modeled data in the XML source are wrapped in a root element. Otherwise it will not work correctly.

Here is an example of using the XML parser.

```
InputStream resourceAsStream == ExampleClass.class.getResourceAsStream("/example-module.
↪yang");

XMLInputFactory factory == XMLInputFactory.newInstance();
```

(continues on next page)

(continued from previous page)

```
XMLStreamReader reader == factory.createXMLStreamReader(resourceAsStream);

NormalizedNodeResult result == new NormalizedNodeResult();
NormalizedNodeStreamWriter streamWriter == ImmutableNormalizedNodeStreamWriter.
↪from(result);

XmlParserStream xmlParser == XmlParserStream.create(streamWriter, schemaContext);
xmlParser.parse(reader);

NormalizedNode<?, ?> transformedInput == result.getResult();
```

The XML parser utilizes the `javax.xml.stream.XMLStreamReader` for parsing an XML document. First, you should create an instance of this reader using XMLInputFactory and then load an XML document (in the form of InputStream object) into it.

In order to emit node events while parsing the data you need to instantiate a NormalizedNodeStreamWriter. This writer is actually an interface and therefore you need to use a concrete implementation of it. In this example it is the ImmutableNormalizedNodeStreamWriter, which constructs immutable instances of NormalizedNodes.

There are two ways how to create an instance of this writer using the static overloaded method from(). One version of this method takes a NormalizedNodeResult as argument. This object type is a result holder in which the resulting NormalizedNode will be stored. The other version takes a NormalizedNodeContainerBuilder as argument. All created nodes will be written to this builder.

Next step is to create an instance of the XML parser. The parser itself is represented by a class named XmlParserStream. You can use one of two versions of the static overloaded method create() to construct this object. One version accepts a NormalizedNodeStreamWriter and a SchemaContext as arguments, the other version takes the same arguments plus a SchemaNode. Node events are emitted to the writer. The SchemaContext is used to check if the YANG data in the XML source comply with the provided YANG model(s). The last argument, a SchemaNode object, describes the node that is the parent of nodes defined in the XML data. If you do not provide this argument, the parser sets the SchemaContext as the parent node.

The parser is now ready to walk through the XML. Parsing is initiated by calling the method parse() on the XmlParserStream object with XMLStreamReader as its argument.

Finally, you can access the result of parsing - a tree of NormalizedNodes contains the data as they are defined in the parsed XML document - by calling the method `getResult()` on the NormalizedNodeResult object.

1.4.3 Project-specific Developer Guides

- InfraUtils Documentation
- JSON RPC Documentation
- OVSDB Documentation
- TransportPCE Documentation

CHAPTER

TWO

CONTRIBUTING TO OPENDAYLIGHT

- *Contributor Guides*
- Infrastructure Guide
- Integration Testing Guide
- Integration Distribution Guide
- Integration Packaging Guide
- *Release Process Guide*
- *Documentation Guide*
- *Javadocs*

CHAPTER

THREE

OPENDAYLIGHT PROJECT DOCUMENTATION

3.1 Managed Projects

- AAA Documentation
- BGPCEP Documentation
- Controller Documentation
- DAEXIM Documentation
- Infrautils Documentation
- JSON RPC Documentation
- LISP Flow Mapping Documentation
- MD-SAL Documentation
- NetConf Documentation
- OpenFlowPlugin Documentation
- OVSDB Documentation
- TransportPCE Documentation

3.2 Self-Managed Projects

3.2.1 Documentation Guide

This guide provides details on how to contribute to the OpenDaylight documentation. OpenDaylight currently uses reStructuredText for documentation and Sphinx to build it. These documentation tools are widely used in open source communities to produce both HTML and PDF documentation and can be easily versioned alongside the code. reStructuredText also offers similar syntax to Markdown, which is familiar to many developers.

Contents

Style Guide

This section serves two purposes:

1. A guide for those writing documentation.
2. A guide for those reviewing documentation.

Note: When reviewing content, assuming that the content is usable, the documentation team is biased toward merging the content rather than blocking it due to relatively minor editorial issues.

Formatting Preferences

In general, when reviewing content, the documentation team ensures that it is comprehensible but tries not to be overly pedantic. Along those lines, while it is preferred that the following formatting preferences are followed, they are generally not an exclusive reason to give a "-1" reply to a patch in Gerrit:

- No trailing whitespace
- Line wrapping at something reasonable, that is, 72–100 characters

Key terms

- Functionality: something useful a project provides abstractly
- Feature: a Karaf feature that somebody could install
- Project: a project within OpenDaylight; projects ship features to provide functionality
- OpenDaylight: this refers to the software we release; use this in place of OpenDaylight controller, the OpenDaylight controller, not ODL, not ODC
  - Since there is a controller project within OpenDaylight, using other terms is hard.

Common writing style mistakes

- In per-project user documentation, you should never say *git clone*, but should assume people have downloaded and installed the controller per the getting started guide and start with `feature:install <something>`
- Avoid statements which are true about part of OpenDaylight, but not generally true.
  - For example: "OpenDaylight is a NETCONF controller." It is, but that is not all it is.
- In general, developer documentation should target external developers to your project so should talk about what APIs you have and how they could use them. It *should not* document how to contribute to your project.

Grammar Preferences

- Avoid contractions: Use "cannot" instead of "can't", "it is" instead of "it's", and so on.

Word Choice

Note: The following word choice guidelines apply when using these terms in text. If these terms are used as part of a URL, class name, or any instance where modifying the case would create issues, use the exact capitalization and spacing associated with the URL or class name.

- ACL: not `Acl` or `acl`
- API: not `api`
- ARP: not `Arp` or `arp`
- datastore: not data store, Data Store, or DataStore (unless it is a class/object name)
- `IPsec`: not IPSEC or `ipsec`
- `IPv4` or `IPv6`: not `Ipv4`, `Ipv6`, `ipv4`, `ipv6`, `IPV4`, or `IPV6`
- Karaf: not karaf
- Linux: not LINUX or `linux`
- NETCONF: not Netconf or netconf
- Neutron: not neutron
- OSGi: not `osgi` or OSGI
- `Open vSwitch`: not OpenvSwitch, OpenVSwitch, or Open V Switch.
- OpenDaylight: not `Opendaylight`, `Open Daylight`, or `OpenDayLight`.

  Note: Also, avoid OpenDaylight abbreviations such as ODL and OD.

- OpenFlow: not `Openflow`, `Open Flow`, or `openflow`.
- OpenStack: not `Open Stack` or `Openstack`
- QoS: not `Qos`, `QOS`, or `qos`
- RESTCONF: not `Restconf` or `restconf`
- RPC not `Rpc` or `rpc`

- URL not `Url` or `url`
- VM: not `Vm` or `vm`
- YANG: not Yang or yang

reStructuredText-based Documentation

When using reStructuredText, follow the Python documentation style guidelines. See: https://devguide.python.org/documenting/

One of the best references for reStructuredText syntax is the Sphinx Primer on reStructuredText.

To build and review the reStructuredText documentation locally, you must have the following packages installed locally:

- python
- python-tox

Note: Both packages should be available in most distribution package managers.

Then simply run `tox` and open the HTML produced by using your favorite web browser as follows:

```
git clone https://git.opendaylight.org/gerrit/docs
cd docs
git submodule update --init
tox
firefox docs/_build/html/index.html
```

Directory Structure

The directory structure for the reStructuredText documentation is rooted in the `docs` directory inside the `docs` `git` repository.

Note: There are guides hosted directly in the `docs` `git` repository and there are guides hosted in remote `git` repositories. Documentation hosted in remote `git` repositories are generally provided for project-specific information.

For example, here is the directory layout on `June, 28th 2016`:

```
$ tree -L 2
.
├── Makefile
├── conf.py
├── documentation.rst
├── getting-started-guide
│   ├── api.rst
│   ├── concepts_and_tools.rst
│   ├── experimental_features.rst
│   ├── index.rst
│   ├── installing_opendaylight.rst
│   ├── introduction.rst
│   ├── karaf_features.rst
```

(continues on next page)

(continued from previous page)

```
    ├── other_features.rst
    ├── overview.rst
    └── who_should_use.rst
├── index.rst
├── make.bat
├── opendaylight-with-openstack
    ├── images
    ├── index.rst
    ├── openstack-with-gbp.rst
    ├── openstack-with-ovsdb.rst
    └── openstack-with-vtn.rst
└── submodules
    └── releng
        └── builder
```

The `getting-started-guide` and `opendaylight-with-openstack` directories correspond to two guides hosted in the `docs` repository, while the `submodules/releng/builder` directory houses documentation for the RelEng/Builder project.

Each guide includes an `index.rst` file, which uses a `toctree` directive that includes the other files associated with the guide. For example:

```
.. toctree::
   :maxdepth: 1

   getting-started-guide/index
   opendaylight-with-openstack/index
   submodules/releng/builder/docs/index
```

This example creates a table of contents on that page where each heading of the table of contents is the root of the files that are included.

Note: When including `.rst` files using the `toctree` directive, omit the `.rst` file extension at the end of the file name.

Adding a submodule

If you want to import a project underneath the documentation project so that the docs can be kept in the separate repository, you can do it by using the `git submodule add` command as follows:

```
git submodule add -b master ../integration/packaging docs/submodules/integration/
↪packaging
git commit -s
```

Note: Most projects will not want to use `-b master`, but instead use the branch `.`, which tracks whatever branch of the documentation project you happen to be on.

Unfortunately, `-b .` does not work, so you have to manually edit the `.gitmodules` file to add `branch = .` and then commit it. For example:

```
<edit the .gitmodules file>
git add .gitmodules
git commit --amend
```

When you're done you should have a git commit something like:

```
$ git show
commit 7943ce2cb41cd9d36ce93ee9003510ce3edd7fa9
Author: Daniel Farrell <dfarrell@redhat.com>
Date:   Fri Dec 23 14:45:44 2016 -0500

    Add Int/Pack to git submodules for RTD generation

    Change-Id: I64cd36ca044b8303cb7fc465b2d91470819a9fe6
    Signed-off-by: Daniel Farrell <dfarrell@redhat.com>

diff --git a/.gitmodules b/.gitmodules
index 91201bf6..b56e11c8 100644
--- a/.gitmodules
+++ b/.gitmodules
@@ -38,3 +38,7 @@
        path = docs/submodules/ovsdb
        url = ../ovsdb
        branch = .
+[submodule "docs/submodules/integration/packaging"]
+       path = docs/submodules/integration/packaging
+       url = ../integration/packaging
+       branch = master
diff --git a/docs/submodules/integration/packaging b/docs/submodules/integration/
↪packaging
new file mode 160000
index 00000000..fd5a8185
--- /dev/null
+++ b/docs/submodules/integration/packaging
@@ -0,0 +1 @@
+Subproject commit fd5a81853e71d45945471d0f91bbdac1a1444386
```

As usual, you can push it to Gerrit with `git review`.

Important: It is critical that the Gerrit patch be merged before the git commit hash of the submodule changes. Otherwise, Gerrit is not able to automatically keep it up-to-date for you.

Documentation Layout and Style

As mentioned previously, OpenDaylight aims to follow the Python documentation style guidelines, which defines a few types of sections:

```
# with overline, for parts
* with overline, for chapters
=, for sections
-, for subsections
^, for subsubsections
", for paragraphs
```

OpenDaylight documentation is organized around the following structure based on that recommendation:

```
docs/index.rst                  -> entry point
docs/____-guide/index.rst       -> part
docs/____-guide/<chapter>.rst   -> chapter
```

In the `____-guide/index.rst` we use the # with overline at the very top of the file to determine that it is a part and then within each chapter file we start the document with a section using * with overline to denote that it is the chapter heading and then everything in the rest of the chapter should use:

```
=, for sections
-, for subsections
^, for subsubsections
", for paragraphs
```

Referencing Sections

This section provides a quick primer for creating references in OpenDaylight documentation. For more information, refer to Cross-referencing documents.

Within a single document, you can reference another section simply by:

```
This is a reference to `The title of a section`_
```

Assuming that somewhere else in the same file, there a is a section title something like:

```
The title of a section
^^^^^^^^^^^^^^^^^^^^^^
```

It is typically better to use `:ref:` syntax and labels to provide links as they work across files and are resilient to sections being renamed. First, you need to create a label something like:

```
.. _a-label:

The title of a section
^^^^^^^^^^^^^^^^^^^^^^
```

Note: The underscore (_) before the label is required.

Then you can reference the section anywhere by simply doing:

```
This is a reference to :ref:`a-label`
```

or:

```
This is a reference to :ref:`a section I really liked <a-label>`
```

Note: When using `:ref:`-style links, you don't need a trailing underscore (_).

Because the labels have to be unique, a best practice is to prefix the labels with the project name to help share the label space; for example, use `sfc-user-guide` instead of just `user-guide`.

Referencing JIRA issues

In order to reference JIRA, we provide two new directives, `jira_fixed_issues` and `jira_known_issues`. These render a table of issues for a particular project and its version range. These are used like this:

```
.. jira_fixed_issues::
   :project: CONTROLLER
   :versions: 4.0.0-4.0.3

.. jira_known_issues::
   :project: CONTROLLER
   :versions: 4.0.0-4.0.3
```

Troubleshooting

Nested formatting does not work

As stated in the reStructuredText guide, inline markup for bold, italic, and fixed-width font cannot be nested. Furthermore, inline markup cannot be mixed with hyperlinks, so you cannot have a link with bold text.

This is tracked in a Docutils FAQ question, but there is no clear current plan to fix this.

Make sure you have cloned submodules

If you see an error like this:

```
./build-integration-robot-libdoc.sh: line 6: cd: submodules/integration/test/csit/
↪libraries: No such file or directory
Resource file '*.robot' does not exist.
```

It means that you have not pulled down the git submodule for the integration/test project. The fastest way to do that is:

```
git submodule update --init
```

In some cases, you might wind up with submodules which are somehow out-of-sync. In that case, the easiest way to fix them is to delete the submodules directory and then re-clone the submodules:

```
rm -rf docs/submodules/
git submodule update --init
```

Warning: These steps delete any local changes or information you made in the submodules, which would only occur if you manually edited files in that directory.

Clear your tox directory and try again

Sometimes, tox will not detect when your `requirements.txt` file has changed and so will try to run things without the correct dependencies. This issue usually manifests as `No module named X` errors or an `ExtensionError` and can be fixed by deleting the `.tox` directory and building again:

```
rm -rf .tox
tox
```

Builds on Read the Docs

Read the Docs builds do not automatically clear the file structure between builds and clones. The result is that you may have to clean up the state of old runs of the build script.

As an example, refer to the following patch: https://git.opendaylight.org/gerrit/c/docs/+/41679/

This patch fixed the issue that caused builds to fail because they were taking too long removing directories associated with generated Javadoc files that were present from previous runs.

Errors from `Coala`

As part of running `tox`, two environments run: `coala` which does a variety of reStructuredText (and other) linting, and `docs`, which runs Sphinx to build HTML and PDF documentation. You can run them independently by doing `tox -ecoala` or `tox -edocs`.

The `coala` linter for reStructuredText is not always the most helpful in explaining why it failed. So, here are some common ones. There should also be Jenkins Failure Cause Management rules that will highlight these for you.

Git Commit Message Errors

`Coala` checks that git commit messages adhere to the following rules:

- `Shortlog` (1st line of commit message) is less than 50 characters
- `Shortlog` (1st line of commit message) is in the imperative mood. For example, "Add foo unit test" is good, but "Adding foo unit test is bad""
- Body (all lines but 1st line of commit message) are less than 72 characters. Some exceptions seem to exist, such as for long URLs.

Some examples of those being logged are:

::

Project wide: | | [NORMAL] GitCommitBear: | | `Shortlog` of HEAD commit isn't in imperative mood! Bad words are 'Adding'

::

Project wide: | | [NORMAL] GitCommitBear: | | Body of HEAD commit contains too long lines. Commit body lines should not exceed 72 characters.

Error in "code-block" directive

If you see an error like this:

::

docs/gerrit.rst | 89| ···..·code-block::·bash | | [MAJOR] RSTcheckBear: | | (ERROR/3) Error in "code-block" directive:

It means that the relevant code-block is not valid for the language specified, in this case `bash`.

Note: If you do not specify a language, the default language is Python. If you want the code-block to not be an any particular language, instead use the `::` directive. For example:

::

::

This is a code block that will not be parsed in any particular language

Project Documentation Requirements

Submitting Documentation Outlines (M2)

1. Determine the features your project will have and which ones will be ''user-facing".
   - In general, a feature is user-facing if it creates functionality that a user would directly interact with.
   - For example, `odl-openflowplugin-flow-services-ui` is likely user-facing since it installs user-facing OpenFlow features, while `odl-openflowplugin-flow-services` is not because it provides only developer-facing features.
2. Determine pieces of documentation that you need to provide based on the features your project will have and which ones will be user-facing.
   - The kinds of required documentation can be found below in the *Requirements for projects* section.

   Note: You might need to create multiple documents for the same kind of documentation. For example, the controller project will likely want to have a developer section for the config subsystem as well as for the MD-SAL.

   ---

3. Clone the docs repository: `git clone https://git.opendaylight.org/gerrit/docs`
4. For each piece of documentation find the corresponding template in the docs repository.
   - For user documentation: `docs.git/docs/templates/template-user-guide.rst`
   - For developer documentation: `ddocs/templates/template-developer-guide.rst`
   - For installation documentation (if any): `docs/templates/template-install-guide.rst`

   Note: You can find the rendered templates here:

<Feature> User Guide

Refer to this template to identify the required sections and information that you should provide for a User Guide. The user guide should contain configuration, administration, management, using, and troubleshooting sections for the feature.

Overview

Provide an overview of the feature and the use case. Also include the audience who will use the feature. For example, audience can be the network administrator, cloud administrator, network engineer, system administrators, and so on.

<Feature> Architecture

Provide information about feature components and how they work together. Also include information about how the feature integrates with OpenDaylight. An architecture diagram could help.

Note: Please *do not* include detailed internals that somebody using the feature wouldn't care about. For example, the fact that there are four layers of APIs between a user command and a message being sent to a device is probably not useful to know unless they have some way to influence how those layers work and a reason to do so.

Configuring <feature>

Describe how to configure the feature or the project after installation. Configuration information could include day-one activities for a project such as configuring users, configuring clients/servers and so on.

Administering or Managing <feature>

Include related command reference or operations that you could perform using the feature. For example viewing network statistics, monitoring the network, generating reports, and so on.

For example:

To configure L2switch components perform the following steps.

(a) Step 1:

(b) Step 2:

(c) Step 3:

Tutorials

optional

If there is only one tutorial, you skip the "Tutorials" section and instead just lead with the single tutorial's name. If you do, also increase the header level by one, i.e., replace the carets (^^^) with dashes (- - -) and the dashes with equals signs (===).

<Tutorial Name>

Ensure that the title starts with a gerund. For example using, monitoring, creating, and so on.

Overview

An overview of the use case.

Prerequisites

Provide any prerequisite information, assumed knowledge, or environment required to execute the use case.

Target Environment

Include any topology requirement for the use case. Ideally, provide visual (abstract) layout of network diagrams and any other useful visual aides.

Instructions

Use case could be a set of configuration procedures. Including screenshots to help demonstrate what is happening is especially useful. Ensure that you specify them separately. For example:

Setting up the VM

To set up a VM perform the following steps.

(a) Step 1

(b) Step 2

(c) Step 3

Installing the feature

To install the feature perform the following steps.

(a) Step 1

(b) Step 2

(c) Step 3

Configuring the environment

To configure the system perform the following steps.

(a) Step 1

(b) Step 2

(c) Step 3

<Feature> Developer Guide

Overview

Provide an overview of the feature, what it logical functionality it provides and why you might use it as a developer. To be clear the target audience for this guide is a developer who will be *using* the feature to build something separate, but *not* somebody who will be developing code for this feature itself.

Note: More so than with user guides, the guide may cover more than one feature. If that is the case, be sure to list all of the features this covers.

<Feature> Architecture

Provide information about feature components and how they work together. Also include information about how the feature integrates with OpenDaylight. An architecture diagram could help. This may be the same as the diagram used in the user guide, but it should likely be less abstract and provide more information that would be applicable to a developer.

Key APIs and Interfaces

Document the key things a user would want to use. For some features, there will only be one logical grouping of APIs. For others there may be more than one grouping.

Assuming the API is MD-SAL- and YANG-based, the APIs will be available both via RESTCONF and via Java APIs. Giving a few examples using each is likely a good idea.

API Group 1

Provide a description of what the API does and some examples of how to use it.

API Group 2

Provide a description of what the API does and some examples of how to use it.

API Reference Documentation

Provide links to JavaDoc, REST API documentation, etc.

<Feature> Installation Guide

Note: *Only* use this template if installation is more complicated than simply installing a feature in the Karaf distribution. Otherwise simply provide the names of all user-facing features in your M3 readout.

This is a template for installing a feature or a project developed in the ODL project. The *feature* could be interfaces, protocol plug-ins, or applications.

Overview

Add overview of the feature. Include Architecture diagram and the positioning of this feature in overall controller architecture. Highlighting the feature in a different color within the overall architecture must help. Include information to describe if the project is within ODL installation package or to be installed separately.

Pre-Requisites for Installing <Feature>

- Hardware Requirements
- Software Requirements

Preparing for Installation

Include any pre-configuration, database, or other software downloads required to install <feature>.

Installing <Feature>

Include if you have separate procedures for Windows and Linux

Verifying your Installation

Describe how to verify the installation.

Troubleshooting

optional

Text goes here.

Post Installation Configuration

Post Installation Configuration section must include some basic (must-do) procedures if any, to get started.

Mandatory instructions to get started with the product.

- Logging in
- Getting Started
- Integration points with controller

Upgrading From a Previous Release

Text goes here.

Uninstalling <Feature>

Text goes here.

---

5. Copy the template into the appropriate directory for your project.
   - For user documentation: `docs.git/docs/user-guide/${feature-name}-user-guide.rst`
   - For developer documentation: `docs.git/docs/developer-guide/${feature-name}-developer-guide.rst`
   - For installation documentation (if any): `docs.git/docs/getting-started-guide/project-specific-guides/${project-name}.rst`

Note: These naming conventions are not set in stone, but are used to maintain a consistent document taxonomy. If these conventions are not appropriate or do not make sense for a document in development, use the convention that you think is more appropriate and the documentation team will review it and give feedback on the Gerrit patch.

---

6. Edit the template to fill in the outline of what you will provide using the suggestions in the template. If you feel like a section is not needed, feel free to omit it.
7. Link the template into the appropriate core `.rst` file.
   - For user documentation: `docs.git/docs/user-guide/index.rst`
   - For developer documentation: `docs.git/docs/developer-guide/index.rst`

- For installation documentation (if any): `docs.git/docs/getting-started-guide/project-specific-guides/index.rst`
- In each file, it should be pretty clear what line you need to add. In general if you have an `.rst` file `project-name.rst`, you include it by adding a new line `project-name` without the `.rst` at the end.

8. Make sure the documentation project still builds.
   - Run `tox` from the root of the cloned docs repository.
     - After that, you should be able to find the HTML version of the docs at `docs.git/docs/_build/html/index.html`.
     - See *reStructuredText-based Documentation* for more details about building the docs.
   - The *reStructuredText Troubleshooting* section provides common errors and solutions.
   - If you still have problems e-mail the documentation group at documentation@lists.opendaylight.org
9. Commit and submit the patch.
   1. Commit using:

      ```
      git add --all && git commit -sm "Documentation outline for ${project-shortname}"
      ```

   2. Submit using:

      ```
      git review
      ```

      See the Git-review Workflow page if you don't have git-review installed.
10. Wait for the patch to be merged or to get feedback
    - If you get feedback, make the requested changes and resubmit the patch.
    - When you resubmit the patch, it is helpful if you also post a "+0" reply to the patch in Gerrit, stating what patch set you just submitted and what you fixed in the patch set.

Expected Output From Documentation Project

The expected output is (at least) 3 PDFs and equivalent web-based documentation:

- User/Operator Guide
- Developer Guide
- Installation Guide

These guides will consist of "front matter" produced by the documentation group and the per-project/per-feature documentation provided by the projects.

Note: This requirement is intended for the person responsible for the documentation and should not be interpreted as preventing people not normally in the documentation group from helping with front matter nor preventing people from the documentation group from helping with per-project/per-feature documentation.

Project Documentation Requirements

Content Types

These are the expected kinds of documentation and target audiences for each kind.

- **User/Operator:** for people looking to use the feature without writing code
  - Should include an overview of the project/feature
  - Should include description of available configuration options and what they do
- **Developer:** for people looking to use the feature in code without modifying it
  - Should include API documentation, such as, enunciate for REST, Javadoc for Java, ??? for REST-CONF/models
- **Contributor:** for people looking to extend or modify the feature's source code

---

Note: You can find this information on the wiki.

- **Installation:** for people looking for instructions to install the feature after they have downloaded the ODL release

---

Note: The audience for this content is the same as User/Operator docs

---

  - For most projects, this will be just a list of top-level features and options
    - As an example, l2switch-switch as the top-level feature with the -rest and `-ui` options
    - Features should also note if the options should be check-boxes (that is, they can each be turned on/off independently) or a drop-down (that is, at most one can be selected)
    - What other top-level features in the release are incompatible with each feature
    - This will likely be presented as a table in the documentation and the data will likely also be consumed by an automated installer or configurator or even downloader.
  - For some projects, there is extra installation instructions (for external components) and/or configuration
    - In that case, there will be a (sub)section in the documentation describing this process.
- **HowTo/Tutorial:** walk-through and examples that are not general-purpose documentation
  - Generally, these should be done as a (sub)section of either user/operator or developer documentation.
  - If they are especially long or complex, they may belong on their own
- **Release Notes:**
  - Release notes are required as part of each project's release review. They must also be translated into re-StructuredText for inclusion in the formal documentation.

Requirements for projects

- Projects must provide reStructuredText documentation including:
  - Developer documentation for every feature
    - Most projects will want to logically nest the documentation for individual features under a single project-wide chapter or section
    - The feature documentation can be provided as a single `.rst` file or multiple `.rst` files if the features fall into different groups
    - Feature documentation should start with approximately 300 words overview of the project and include references to any automatically-generated API documentation as well as more general developer information (see *Content Types*).
  - User/Operator documentation for every every user-facing feature (if any)
    - This documentation should be per-feature, not per-project. Users should not have to know which project a feature came from.
    - Intimately related features can be documented together. For example, `l2switch-switch`, `l2switch-switch-rest`, and `l2switch-switch-ui`, can be documented as one noting the differences.
    - This documentation can be provided as a single `.rst` file or multiple `.rst` files if the features fall into different groups
  - Installation documentation
    - Most projects will simply provide a list of user-facing features and options. See *Content Types* above.
  - Release Notes (both on the wiki and reStructuredText) as part of the release review.
- Documentation must be imported from the project own repository or contributed to the docs repository for generic information.
  - Projects may be encouraged to instead provide this from their own repository if the tooling is developed
  - Projects choosing to meet the requirement in this way must provide a patch to docs repository to import the project's documentation
- Projects must cooperate with the documentation group on edits and enhancements to documentation

Timeline for Deliverables from Projects

- **M2:** Documentation Started

  The following tasks for documentation deliverables must be completed for the M2 readout:

  - The kinds of documentation that will be provided and for what features must be identified.

    Note: Release Notes are not required until release reviews at **RC2**

  - The appropriate `.rst` files must be created in the docs repository (or their own repository if the tooling is available).
  - An outline for the expected documentation must be completed in those `.rst` files including the relevant (sub)sections and a sentence or two explaining what will be contained in these sections.

> **Note:** If an outline is not provided, delivering actual documentation in the (sub)sections meets this requirement.

  - M2 readouts should include
    1. the list of kinds of documentation
    2. the list of corresponding `.rst` files and their location, including repository and path
    3. the list of commits creating those `.rst` files
    4. the current word counts of those `.rst` files
- **M3:** Documentation Continues
  - The readout at M3 should include the word counts of all `.rst` files with links to commits
  - The goal is to have draft documentation complete at the M3 readout so that the documentation group can comment on it.
- **M4:** Documentation Complete
  - All (sub)sections in all `.rst` files have complete, readable, usable content.
  - Ideally, there should have been some interaction with the documentation group about any suggested edits and enhancements
- **RC2:** Release notes
  - Projects must provide release notes in `.rst` format pushed to integration (or locally in the project's repository if the tooling is developed)

3.2.2 OpenDaylight Release Process Guide

Overview

This guide provides details on the various release processes related to OpenDaylight. It documents the steps used by OpenDaylight release engineers when performing release operations.

Release Planning

Managed Release

Managed Release Summary

The Managed Release Process will facilitate timely, stable OpenDaylight releases by allowing the release team to focus on closely managing a small set of core OpenDaylight projects while not imposing undue requirements on projects that prefer more autonomy.

Managed Release Goals

Reduce Overhead on Release Team

The Managed Release Model will allow the release team to focus their efforts on a smaller set of more stable, more responsive projects.

Reduce Overhead on Projects

The Managed Release Model will reduce the overhead both on projects taking part in the Managed Release and Self-Managed Projects.

Managed Projects will have fewer, smaller checkpoints consisting of only information that is maximally helpful for driving the release process. Much of the information collected at checkpoints will be automatically scraped, requiring minimal to no effort from projects. Additionally, Managed Release projects should have a more stable development environment, as the projects that can break the jobs they depend on will be a smaller set, more stable and more responsive.

Projects that are Self-Managed will have less overhead and reporting. They will be free to develop in their own way, providing their artifacts to include in the final release or choosing to release on their own schedule. They will not be required to submit any checkpoints and will not be expected to work closely with the rest of the OpenDaylight community to resolve breakages.

Enable Timely Releases

The Managed Release Process will reduce the set of projects that must simultaneously become stable at release time. The release and test teams will be able to focus on orchestrating a quality release for a smaller set of more stable, more responsive projects. The release team will also have greater latitude to step in and help projects that are required for dependency reasons but may not have a large set of active contributors.

Managed Projects

Managed Projects Summary

Managed Projects are either required by most applications for dependency reasons or are mature, stable, responsive projects that are consistently able to take part in releases without jeopardizing them. Managed Projects will receive additional support from the test and release teams to further their stability and make sure OpenDaylight releases go out on-time. To enable this extra support, Managed Projects will be given less autonomy than OpenDaylight projects have historically been granted.

Managed Projects for Dependency Reasons

Some projects are required by almost all other OpenDaylight projects. These projects must be in the Managed Release for it to support almost every OpenDaylight use case. Such projects will not have a choice about being in the Managed Release, the TSC will decide they are critical to the OpenDaylight platform and include them. They may not always meet the requirements that would normally be imposed on projects that wish to join the Managed Release. Since they cannot be kicked out of the release, the TSC, test and release teams will do their best to help them meet the Managed Release Requirements. This may involve giving Linux Foundation staff temporary committer rights to merge patches on behalf of unresponsive projects, or appointing committers if projects continue to remain unresponsive. The TSC will

strive to work with projects and member companies to proactively keep projects healthy and find active contributors who can become committers in the normal way without the need to appoint them in times of crisis.

Managed Release Integrated Projects

Some Managed Projects may decide to release on their own, not as a part of the Simultaneous Release with Snapshot Integrated Projects. Such Release Integrated projects will still be subject to Managed Release Requirements, but will need to follow a different release process.

For implementation reasons, the projects that are able to release independently must depend only on other projects that release independently. Therefore the Release Integrated Projects will form a tree starting from odlparent. Currently the following projects are Release Integrated:

- aaa
- controller
- infrautils
- mdsal
- netconf
- odlparent
- yangtools

Requirements for Managed Projects

Healthy Community

Managed Projects should strive to have a healthy community.

Responsiveness

Managed Projects should be responsive over email, IRC, Gerrit, Jira and in regular meetings.

All committers should be subscribed to their project's mailing list and the release mailing list.

For the following particularly time-sensitive events, an appropriate response is expected within two business days.

- RC or SR candidate feedback.
- Major disruptions to other projects where a Jira weather item was not present and the pending breakage was not reported to the release mailing list.

If anyone feels that a Managed Project is not responsive, a grievance process is in place to clearly handle the situation and keep a record for future consideration by the TSC.

Active Committers

Managed Projects should have sufficient active committers to review contributions in a timely manner, support potential contributors, keep CSIT healthy and generally effectively drive the project.

If a project that the TSC deems is critical to the Managed Release is shown to not have sufficient active committers the TSC may step in and appoint additional committers. Projects that can be dropped from the Managed Release will be dropped instead of having additional committers appointed.

Managed Projects should regularly prune their committer list to remove inactive committers, following the Committer Removal Process.

TSC Attendance

Managed Projects are required to send a representative to attend TSC meetings.

To facilitate quickly acting on problems identified during TSC meetings, representatives must be a committer to the project they are representing. A single person can represent any number of projects.

Representatives will make the following entry into the meeting minutes to record their presence:

#project <project ID>

TSC minutes will be scraped per-release to gather attendance statistics. If a project does not provide a representative for at least half of TSC meetings a grievance will be filed for future consideration.

Checkpoints Submitted On-Time

Managed Projects must submit information required for checkpoints on-time. Submissions must be correct and adequate, as judged by the release team and the TSC. Inadequate or missing submissions will result in a grievance.

Jobs Required for Managed Projects Running

Managed Projects are required to have the following jobs running and healthy.

- Distribution check job (voting)
- Validate autorelease job (voting)
- Merge job (non-voting)
- Sonar job (non-voting)
- CLM job (non-voting)

Depend only on Managed Projects

Managed Projects should only depend on other Managed Projects.

If a project wants to be Managed but depends on Self-Managed Projects, they should work with their dependencies to become Managed at the same time or drop any Self-Managed dependencies.

Documentation

Managed Projects are required to produce a user guide, developer guide and release notes for each release.

CLM

Managed Projects are required to handle CLM (Component Life-cycle Management) violations in a timely manner.

Managed Release Process

Managed Release Checkpoints

Checkpoints are designed to be mostly automated, to be maximally effective at driving the release process and to impose as little overhead on projects as possible.

There will be an initial checkpoint two weeks after the start of the release, a midway checkpoints one month before code freeze and a final checkpoint at code freeze.

Initial Checkpoint

An initial checkpoint will be collected two weeks after the start of each release. The release team will review the information collected and report it to the TSC at the next TSC meeting.

Projects will need to create the following artifacts:

- High-level, human-readable description of what the project plans to do in this release. This should be submitted as a Jira **Project Plan** issue against the TSC project.
  - Select your project in the **ODL Project** field
  - Select the release version in the **ODL Release** field
  - Select the appropriate value in the **ODL Participation** field: `SNAPSHOT_Integrated (Managed)` or `RELEASE_Integrated (Managed)`
  - Select the value `Initial` in the **ODL Checkpoint** field
  - In the **Summary** field, put something like: `Project-X Fluorine Release Plan`
  - In the **Description** field, fill in the details of your plan:

```
This should list a high-level, human-readable summary of what a project
plans to do in a release. It should cover the project's planned major
accomplishments during the release, such as features, bugfixes, scale,
stability or longevity improvements, additional test coverage, better
documentation or other improvements. It may cover challenges the project
is facing and needs help with from other projects, the TSC or the LFN
umbrella. It should be written in a way that makes it amenable to use
for external communication, such as marketing to users or a report to
other LFN projects or the LFN Board.
```

- If a project is transitioning from Self-Managed to Managed or applying for the first time into a Managed release, raise a Jira **Project Plan** issue against the TSC project highlighting the request.
  - Select your project in the **ODL Project** field

- Select the release version in the **ODL Release** field
- Select the `NOT_Integrated (Self-Managed)` value in the **ODL Participation** field
- Select the appropriate value in the **ODL New Participation** field: `SNAPSHOT_Integrated (Managed)` or `RELEASE_Integrated (Managed)`
- In the **Summary** field, put something like: `Project-X joining/moving to Managed Release for Fluorine`
- In the **Description** field, fill in the details using the template below:

```
Summary
This is an example of a request for a project to move from Self-Managed
to Managed. It should be submitted no later than the start of the
release. The request should make it clear that the requesting project
meets all of the Managed Release Requirements.

Healthy Community
The request should make it clear that the requesting project has a
healthy community. The request may also highlight a history of having a
healthy community.

Responsiveness
The request should make it clear that the requesting project is
responsive over email, IRC, Jira and in regular meetings. All committers
should be subscribed to the project's mailing list and the release
mailing list. The request may also highlight a history of
responsiveness.

Active Committers
The request should make it clear that the requesting project has a
sufficient number of active committers to review contributions in a
timely manner, support potential contributors, keep CSIT healthy and
generally effectively drive the project. The requesting project should
also make it clear that they have pruned any inactive committers. The
request may also highlight a history of having sufficient active
committers and few inactive committers.

TSC Attendance
The request should acknowledge that the requesting project is required
to send a committer to represent the project to at least 50% of TSC
meetings. The request may also highlight a history of sending
representatives to attend TSC meetings.

Checkpoints Submitted On-Time
The request should acknowledge that the requesting project is required
to submit checkpoints on time. The request may also highlight a history
of providing deliverables on time.

Jobs Required for Managed Projects Running
The request should show that the requesting project has the required
jobs for Managed Projects running and healthy. Links should be provided.

Depend only on Managed Projects
```

(continues on next page)

(continued from previous page)

```
The request should show that the requesting project only depends on
Managed Projects.

Documentation
The request should acknowledge that the requesting project is required
to produce a user guide, developer guide and release notes for each
release. The request may also highlight a history of providing quality
documentation.

CLM
The request should acknowledge that the requesting project is required
to handle Component Lifecycle Violations in a timely manner. The request
should show that the project's CLM job is currently healthy. The request
may also show that the project has a history of dealing with CLM
violations in a timely manner.
```

- If a project is transitioning from Managed to Self-Managed, raise a Jira **Project Plan** issue against the TSC project highlighting the request.
  - Select your project in the **ODL Project** field
  - Select the release version in the **ODL Release** field
  - Select the appropriate value in the **ODL Participation** field: `SNAPSHOT_Integrated (Managed)` or `RELEASE_Integrated (Managed)`
  - Select the `NOT_Integrated (Self-Managed)` value in the **ODL New Participation** field
  - In the **Summary** field, put something like: `Project-X Fluorine Joining/Moving to Self-Manged for Fluorine`
  - In the **Description** field, fill in the details:

```
This is a request for a project to move from Managed to Self-Managed. It
should be submitted no later than the start of the release. The request
does not require any additional information, but it may be helpful for
the requesting project to provide some background and their reasoning.
```

- Weather items that may impact other projects should be submitted as Jira issues. For a weather item, raise a Jira **Weather Item** issue against the TSC project highlighting the details.
  - Select your project in the **ODL Project** field
  - Select the release version in the **ODL Release** field
  - For the **ODL Impacted Projects** field, fill in the impacted projects using label values - each label value should correspond to the respective project prefix in Jira, e.g. netvirt is NETVIRT. If all projects are impacted, use the label value `ALL`.
  - Fill in the expected date of weather event in the **ODL Expected Date** field
  - Select the appropriate value for **ODL Checkpoint** (may skip)
  - In the **Summary** field, summarize the weather event
  - In the **Description** field, provide the details of the weather event. Provide as much relevant information as possible.

The remaining artifacts will be automatically scraped:

- Blocker bugs that were raised between the previous code freeze and release.
- Grievances raised against the project during the last release.

Midway Checkpoint

One month before code freeze, a midway checkpoint will be collected. The release team will review the information collected and report it to the TSC at the next TSC meeting. All information for midway checkpoint will be automatically collected.

- Open Jira bugs marked as blockers.
- Open Jira issues tracking weather items.
- Statistics about jobs. * Autorelease failures per-project. * CLM violations.
- Grievances raised against the project since the last checkpoint.

Since the midway checkpoint is fully automated, the release team may collect this information more frequently, to provide trends over time.

Final Checkpoint

At 2 weeks after code freeze a final checkpoint will be collected by the release team and presented to the TSC at the next TSC meeting.

Projects will need to create the following artifacts:

- High-level, human-readable description of what the project did in this release. This should be submitted as a Jira **Project Plan** issue against the TSC project. This will be reused for external communication/marketing for the release.
  - Select your project in the **ODL Project** field
  - Select the release version in the **ODL Release** field
  - Select the appropriate value in the **ODL Participation** field: `SNAPSHOT_Integrated (Managed)` or `RELEASE_Integrated (Managed)`
  - Select the value `Final` in the **ODL Checkpoint** field
  - In the **Summary** field, put something like: `Project-X Fluorine Release details`
  - In the **Description** field, fill in the details of your accomplishments:

```
This should be a high-level, human-readable summary of what a project
did during a release. It should cover the project's major
accomplishments, such as features, bugfixes, scale, stability or
longevity improvements, additional test coverage, better documentation
or other improvements. It may cover challenges the project has faced
and needs help in the future from other projects, the TSC or the LFN
umbrella. It should be written in a way that makes it amenable to use
for external communication, such as marketing to users or a report to
other LFN projects or the LFN Board.
```

  - In the **ODL Gerrit Patch** field, fill in the Gerrit patch URL to your project release notes
- Release notes, user guide, developer guide submitted to the docs project.

The remaining artifacts will be automatically scraped:

- Open Jira bugs marked as blockers.
- Open Jira issues tracking weather items.
- Statistics about jobs. * Autorelease failures per-project.
- Statistics about patches. * Number of patches submitted during the release. * Number of patches merged during the release. * Number of reviews per-reviewer.
- Grievances raised against the project since the start of the release.

Service Release Code Freeze

There will be an additional checkpoint for Service Releases, where code will freeze one week before the schedule release. Here are more details:

- After code-freeze, the specific branch will be locked down, just like with a major release.
- Once the branch is locked, only fixes for blocker bugs will be allowed as long as they are approved by TSC and Release PM.
- Bugs or Regression discovered during RC test will also be considered by Release PM.
- Release PM will track and allow those fixes that have not been reviewed or merged because of project low activity or lack of committers. The deadline to communicate the list of patches waiting review for a particular Service Release is the code freeze date.

Managed Release Integrated Release Process

Managed Projects that release independently (Release Integrated Projects), not as a part of the Simultaneous Release with Snapshot Integrated Projects, will need to follow a different release process.

Managed Release Integrated (MRI) Projects will provide the releases they want the Managed Snapshot Integrated (MSI) Projects to consume no later than two weeks after the start of the Managed Release. The TSC will decide by a majority vote whether to bump MSI versions to consume the new MRI releases. This should happen as early in the release as possible to get integration woes out of the way and allow projects to focus on developing against a stable base. If the TSC decide to consume the proposed MRI releases, all MSI Projects are required to bump to the new versions within a two day window. If some projects fail to merge version bump patches in time, the TSC will instruct Linux Foundation staff to temporarily wield committer rights and merge version bump patches. The TSC vote at any time to back out of a version bump if the new releases are found to be unsuitable.

MRI Projects are expected to provide bug fixes via minor or patch version updates during the release, but should strive to not expect MSI Projects to consume another major version update during the release.

MRI Projects are free to follow their own release cadence as they develop new features during the Managed Release. They need only have a stable version ready for the MSI Projects to consume by the next integration point.

Managed Release Integrated Checkpoints

The MRI Projects will follow similar checkpoints as the MSI Projects, but the timing will be different. At the time MRI Projects provide the releases they wish MSI Projects to consume for the next release, they will also provide their final checkpoints. Their midway checkpoints will be scraped one month before the deadline for them to deliver their artifacts to the MSI Projects. Their initial checkpoints will be due no later two weeks following the deadline for their delivery of artifacts to the MSI Projects. Their initial checkpoints will cover everything they expect to do in the next Managed Release, which may encompass any number of major version bumps for the MRI Projects.

Moving a Project from Self-Managed to Managed

Self-Managed Projects can request to become Managed by submitting a **Project_Plan** issue to the TSC project in Jira. See details as described under the *Initial Checkpoint* section above. Requests should be submitted before the start of a release. The requesting project should make it clear that they meet the Managed Release Project Requirements.

The TSC will evaluate requests to become Managed and inform projects of the result and the TSC's reasoning no later than the start of the release or one week after the request was submitted, whichever comes last.

For the first release, the TSC will bootstrap the Managed Release with projects that are critical to the OpenDaylight platform. Other projects will need to follow the normal application process defined above.

The following projects are deemed critical to the OpenDaylight platform:

- aaa
- controller
- infrautils
- mdsal
- netconf
- odlparent
- yangtools

Self-Managed Projects

In general there are two types of Self-Managed (SM) projects:

1. Self-Managed projects that want to participate in the formal (major or service) OpenDaylight release distribution. This section includes the requirements and release process for these projects.
2. Self-Managed projects that want to manage their own release schedule or provide their release distribution and installation instructions by the time of the release. There are no specific requirements for these projects.

Requirements for SM projects participating in the release distribution

Use of SNAPSHOT versions

Self-Managed Projects can consume whichever version of their upstream dependencies they want during most of the release cycle, but if they want to be included in the formal (major or service) release distribution they must have their upstream versions bumped to SNAPSHOT and build successfully no later than one week before the first Managed release candidate (RC) is created. Since bumping and integrating with upstream takes time, it is strongly recommended Self-Managed projects start this work early enough. This is no later than the middle checkpoint if they want to be in a

major release, or by the previous release if they want to be in a service release (e.g. by the major release date if they want to be in SR1).

Note: To help with the integration effort, the Weather Page includes API and other important changes during the release cycle. After the formal release, the release notes also include this information.

Add to Common Distribution

In order to be included in the formal (major or service) release distribution, Self-Managed Projects must be in the common distribution pom.xml file and the distribution sanity test (see Add Projects to distribution) no later than one week before the first Managed release candidate (RC) is created. Projects should only be added to the final distribution pom.xml after they have successfully published artifacts using upstream SNAPSHOTs. See *Use of SNAPSHOT versions*.

Note: It is very important Self-Managed projects do not miss the deadlines for upstream integration and final distribution check, otherwise there are high chances for missing the formal release distribution. See *Release the project artifacts*.

Cut Stable Branch

Self-Managed projects wanting to use the existing release job to release their artifacts (see *Release the project artifacts*) must have an stable branch in the major release (fluorine, neon, etc) they are targeting. It is highly recommended to cut the stable branch before the first Managed release candidate (RC) is created.

After creating the stable branch Self-Managed projects should:

- Bump master branch version to X.Y+1.0-SNAPSHOT, this way any new merge in master will not interfere with the new created stable branch artifacts.
- Update `.gitreview` for stable branch: change `defaultbranch=master` to stable branch. This way folks running "git review" will get the right branch.
- Update their jenkins jobs: current release should point to the new created stable branch and next release should point to master branch. If you do not know how to do this please open a ticket to OpenDaylight helpdesk.

Release the project artifacts

Self-Managed projects wanting to participate in the formal (major or service) release distribution must release and publish their artifacts to nexus in the week after the Managed release is published to nexus.

Self-Managed projects having an stable branch with latest upstream SNAPSHOT (see previous requirements) can use the release job in *Project Standalone Release* to release their artifacts.

After creating the release, Self-Managed projects should bump the stable branch version to X.Y.Z+1-SNAPSHOT, this way any new merge in the stable branch will not interfere with pre-release artifacts.

Note: Self-Managed Projects will not have any leeway for missing deadlines. If projects are not in the final distribution in the allocated time (normally one week) after the Managed projects release, they will not be included in the release distribution.

Checkpoints

There are no checkpoints for Self-Managed Projects.

Moving a Project from Managed to Self-Managed

Managed Projects that are not required for dependency reasons can submit a **Project_Plan** issue to be Self-Managed to the TSC project in Jira. See details in the *Initial Checkpoint* section above. Requests should be submitted before the start of a release. Requests will be evaluated by the TSC.

The TSC may withdraw a project from the Managed Release at any time.

Installing Features from Self-Managed Projects

Self-Managed Projects will have their artifacts included in the final release if they are available on-time, but they will not be available to be installed until the user does a *repo:add* `.

To install an Self-Managed Project feature, find the feature description in the system directory. For example, NetVirt's main feature:

```
system/org/opendaylight/netvirt/odl-netvirt-openstack/0.6.0-SNAPSHOT/
```

Then use the Karaf shell to `repo:add` the feature:

```
feature:repo-add mvn:org.opendaylight.netvirt/odl-netvirt-openstack/0.6.0-SNAPSHOT/xml/
↪features
```

Grievances

For requirements that are difficult to automatically ascertain if a Managed Project is following or not, there should be a clear reporting process.

Grievance reports should be filed against the TSC project in Jira. Very urgent grievances can additionally be brought to the TSC's attention via the TSC's mailing list.

Process for Reporting Unresponsive Projects

If a Managed Project does not meet the *Responsiveness* Requirements, a **Grievance** issue should be filed against the TSC project in Jira.

Unresponsive project reports should include (at least):

- Select the project being reported in the **ODL_Project** field
- Select the release version in the **ODL_Release** field
- In the **Summary** field, put something like: `Grievance against Project-X`
- In the **Description** field, fill in the details:

```
Document the details that show ExampleProject was slow to review a change.
The report should include as much relevant information as possible,
including a description of the situation, relevant Gerrit change IDs and
relevant public email list threads.
```

- In the **ODL_Gerrit_Patch**, put in a URL to a Gerrit patch, if applicable

Vocabulary Reference

- Managed Release Process: The release process described in this document.
- Managed Project: A project taking part in the Managed Release Process.
- Self-Managed Project: A project not taking part in the Managed Release Process.
- Simultaneous Release: Event wherein all Snapshot Integrated Project versions are rewritten to release versions and release artifacts are produced.
- Snapshot Integrated Project: Project that integrates with OpenDaylight projects using snapshot version numbers. These projects release together in the Simultaneous Release.
- Release Integrated Project: Project that releases independently of the Simultaneous Release. These projects are consumed by Snapshot Integrated Projects based on release version numbers, not snapshot versions.

Release Schedule

OpenDaylight releases twice per year. The six-month cadence is designed to synchronize OpenDaylight releases with OpenStack and OPNFV releases. Dates are adjusted to match current resources and requirements from the current OpenDaylight users. Dates are also adjusted when they conflict with holidays, overlap with other releases or are otherwise problematic. Dates are the release deadlines intended for managed projects. Self-managed projects that wants to integrate the distribution have one week to release once managed projects are ready.

Each milestone is usually evaluated at a TSC meeting, which dictates deadlines. Nominal deadline is midnight UTC on the particular date. As an example, a date of 2022-10-06 means that all deliverables are due no later than 2022-10-06T00:00:00Z. The corresponding TSC meeting happens either on 9am or 10pm Pacific time. The slack time between the deadline and the actual call can be used for justified last-minute work if the need arises (though in general WE SHOULD NEVER need it). If such last-minute work results in a SimRel candidate build not being available, the reasons for that need to be discussed at, and documented as part of, the corresponding TSC meeting.

Event	Argon Dates	Chlorine Dates	Sulfur Dates	Relative Dates	Start-Relative Dates	Description
Release Start	2022-09-22	2022-03-17	2021-09-23	Start Date	Start Date +0	Declare Intention: Submit **Project_Plan** Jira item in TSC project.
Initial Checkpoint	2022-10-06	2022-03-31	2021-10-07	Start Date +2 weeks	Start Date +2 weeks	Initial Checkpoint. All Managed Projects must have completed **Project_Plan** Jira items in TSC project.
Release Integrated Deadline	2022-10-20	2022-04-14	2021-10-21	Initial Checkpoint + 2 weeks	Start Date +4 weeks	Deadline for Release Integrated Projects (currently, ODLPARENT, YANGTOOLS, MDSAL, CONTROLLER and INFRAUTILS) to provide the desired version deliverables for downstream Snapshot Integrated Projects to consume.
Version Bump	2022-10-21	2022-04-15	2021-10-22	Release Integrated Deadline + 1 day	Start Date +4 weeks 1 day	Prepare version bump patches and merge them in (RelEng team). Spend the next 2 weeks to get green build for all MSI Projects and a healthy distribution.
Version Bump Checkpoint	2021-11-03	2022-04-29	2021-11-04	Release Integrated Deadline + 2 weeks	Start Date +6 weeks	Check status of MSI Projects to see if we have green builds and a healthy distribution. Revert the MRI deliverables if deemed necessary.
CSIT Checkpoint	2022-11-17	2022-05-13	2021-11-18	Version Bump Checkpoint + 2 weeks	Start Date +8 weeks	All Managed Release CSIT should be in good shape - get all MSI Projects' CSIT results as they were before the version bump. This is the final opportunity to revert the MRI deliverables if deemed necessary.
Middle Checkpoint	2023-01-12	2022-07-07	2022-01-13	CSIT Checkpoint + 8 weeks (sometimes +2 weeks to avoid December holidays)	Start Date +16 weeks (sometimes +2 weeks to avoid December holidays)	Checkpoint for status of Managed Projects - especially Snapshot Integrated Projects.
Code Freeze	2023-02-09	2022-08-04	2022-02-10	Middle Checkpoint + 4 weeks	Start Date +20 weeks	Code freeze for all Managed Projects - cut and lock release branch. Only allow blocker bug fixes in release branch.
Final Checkpoint	2023-02-23	2022-08-19	2022-02-24	Code Freeze + 2 weeks	Start Date +22 weeks	Final Checkpoint for all Managed Projects.
Formal	2023-03-16	2022-09-19	2022-03-17	6 months	Start Date	Formal Release for

Processes

Project Standalone Release

This page explains how a project can release independently outside of the OpenDaylight simultaneous release.

Preparing your project for release

A project can produce a staging repository by using one of the following methods against the {project-name}-maven-stage-{stream} job:

- Leave a comment `stage-release` against any patch for the stream to build
- Click `Build with Parameters` in Jenkins Web UI for the job

This job performs the following duties:

1. Removes -SNAPSHOT from all pom files
2. Produces a `taglist.log`, project.patch, and project.bundle files
3. Runs a *mvn clean deploy* to a local staging repository
4. Pushes the staging repository to a Nexus staging repository https://nexus.opendaylight.org/content/repositories/<REPO_ID> (REPO_ID is saved to `staging-repo.txt` on the log server)
5. Archives `taglist.log`, project.patch, and project.bundle files to log server

The files *`taglist.log`* and *`project.bundle`* can be used later at release time to reproduce a byte exact commit of what was built by the Jenkins job. This can be used to tag the release at release time.

Releasing your project

Once testing against the staging repository has been completed and project has determined that the staged repository is ready for release. A release can the be performed using the self-serve release process: https://docs.releng.linuxfoundation.org/projects/global-jjb/en/latest/jjb/lf-release-jobs.html

1. Ask helpdesk the necessary right on jenkins if you do not have them
2. Log on https://jenkins.opendaylight.org/releng/
3. Choose your project dashboard
4. Check your release branch has been successfully staged and note the corresponding log folder
5. Go back to the dashboard and choose the release-merge job
6. Click on build with parameters
7. Fill in the form:

- GERRIT_BRANCH must be changed to the branch name you want to release (e.g. stable/sodium)
- VERSION with your corresponding project version (e.g. 0.4.1)
- LOG_DIR with the relative path of the log from the stage release job (e.g. project-maven-stage-master/17/)
- choose maven DISTRIBUTION_TYPE in the select box
- uncheck USE_RELEASE_FILE box

8. Launch the jenkins job

This job performs the following duties: * download and patch your project repository * build the project * publish the artifacts on nexus * tag and sign the release on Gerrit

Autorelease

The Release Engineering - Autorelease project is targeted at building the artifacts that are used in the release candidates and final full release.

- Open Gerrit Patches
- Jenkins Jobs

Cloning Autorelease

To clone all the autorelease repository including it's `submodules` simply run the clone command with the '"–recursive"' parameter.

```
git clone --recursive https://git.opendaylight.org/gerrit/releng/autorelease
```

If you forgot to add the –recursive parameter to your git clone you can pull the `submodules` after with the following commands.

```
git submodule init
git submodule update
```

Creating Autorelease - Release and RC build

An autorelease release build comes from the autorelease-release-<branch> job which can be found on the autorelease tab in the releng master:

- https://jenkins.opendaylight.org/releng/view/autorelease/

For example to create a Boron release candidate build launch a build from the autorelease-release-boron job by clicking the '"Build with Parameters"' button on the left hand menu:

- https://jenkins.opendaylight.org/releng/view/autorelease/job/autorelease-release-boron/

Note: The only field that needs to be filled in is the '"RELEASE_TAG"', leave all other fields to their default setting. Set this to Boron, Boron-RC0, Boron-RC1, etc… depending on the build you'd like to create.

Adding Autorelease staging repository to settings.xml

If you are building or testing this release in such a way that requires pulling some of the artifacts from the Nexus repository you may need to modify your `settings.xml` to include the staging repository URL as this URL is not part of OpenDaylight Nexus' public or snapshot groups. If you've already cloned the recommended settings.xml for building ODL you will need to add an additional profile and activate it by adding these sections to the \<`profiles`\> and \<`activeProfiles`\> sections (please adjust accordingly).

Note:

- This is an example and you need to "Add" these example sections to your `settings.xml` do not delete your existing sections.
- The URLs in the \<repository\> and \<pluginRepository\> sections will also need to be updated with the staging repository you want to test.

```
<profiles>
  <profile>
    <id>opendaylight-staging</id>
    <repositories>
      <repository>
        <id>opendaylight-staging</id>
        <name>opendaylight-staging</name>
        <url>https://nexus.opendaylight.org/content/repositories/automatedweeklyreleases-
↪1062</url>
        <releases>
          <enabled>true</enabled>
          <updatePolicy>never</updatePolicy>
        </releases>
        <snapshots>
          <enabled>false</enabled>
        </snapshots>
      </repository>
    </repositories>
    <pluginRepositories>
      <pluginRepository>
        <id>opendaylight-staging</id>
        <name>opendaylight-staging</name>
        <url>https://nexus.opendaylight.org/content/repositories/automatedweeklyreleases-
↪1062</url>
        <releases>
          <enabled>true</enabled>
          <updatePolicy>never</updatePolicy>
        </releases>
        <snapshots>
          <enabled>false</enabled>
        </snapshots>
      </pluginRepository>
    </pluginRepositories>
  </profile>
</profiles>

<activeProfiles>
  <activeProfile>opendaylight-staging</activeProfile>
</activeProfiles>
```

Project life-cycle

This page documents the current rules to follow when adding and removing a particular project to Simultaneous Release (SR).

List of states for projects in autorelease

The state names are short negative phrases describing what is missing to progress to the following state.

- **non-existent** The project is not recognized by Technical Steering Committee (TSC) to be part of OpenDaylight (ODL).
- **non-participating** The project is recognized by the TSC to be an ODL project, but the project has not confirmed participation in SR for given release cycle.
- **non-building** The recognized project is willing to participate, but its current codebase is not passing its own merge job, or the project artifacts are otherwise unavailable in Nexus.
- **not-in-autorelease** Project merge job passes, but the project is not added to autorelease (git submodule, maven module, validate-autorelease job passes).
- **failing-autorelease** The project is added to autorelease (git submodule, maven module, validate-autorelease job passes), but autorelease build fails when building project's artifact. Temporary state, timing out into not-in-autorelease.
- **repo-not-in-integration** Project is successfully built within autorelease, but integration/distribution:features-index is not listing all its public feature repositories.
- **feature-not-in-integration** Feature repositories are referenced, distribution-check job is passing, but some user-facing features are absent from integration/distribution:features-test (possibly because adding them does not pass distribution SingleFeatureTest).
- **distribution-check-not-passing** Features are in distribution, but distribution-check job is either not running, or it is failing for any reason. Temporary state, timing out into feature-not-in-integration.
- **feature-is-experimental** All user-facing features are in features-test, but at least one of the corresponding functional CSIT jobs does not meet Integration/Test requirements.
- **feature-is-not-stable** Feature does meet Integration/Test requirements, but it does not meed all requirements for stable features.
- **feature-is-stable**

Note: A project may change its state in both directions, this list is to make sure a project is not left in an invalid state, for example distribution referencing feature repositories, but without passing distribution-check job.

Note: Projects can participate in Simultaneous Release even if they are not included in autorelease. Nitrogen example: Odlparent. FIXME: Clarify states for such projects (per version, if they released multiple times within the same cycle).

Branch Cutting

This page documents the current branch cutting tasks that are needed to be performed at RC0 and which team has the necessary permissions in order to perform the necessary task in Parentheses.

JJB (releng/builder)

1. Export `${NEXT_RELEASE}` and `${CURR_RELEASE}` with new and current release names. **(releng/builder committers)**

```
export CURR_RELEASE="Silicon"
export NEXT_RELEASE="Phosphorus"
```

2. Run the script `cut-branch-jobs.py` to generate next release jobs. **(releng/builder committers)**

```
python scripts/cut-branch-jobs.py $CURR_RELEASE $NEXT_RELEASE jjb/
pre-commit run --all-files
```

Note: `pre-commit` is necessary to adjust the formatting of the generated YAML.

This script changes JJB yaml files to insert the next release configuration by updating streams and branches where relevant. For example if `master` is currently Silicon, the result of this script will update config blocks as follows:

Update multi-streams:

```
stream:
  - Phosphorus:
      branch: master
  - Silicon:
      branch: stable/silicon
```

Insert project new blocks:

```
- project:
    name: aaa-phosphorus
    jobs:
      - '{project-name}-verify-{stream}-{maven}-{jdks}'
    stream: phosphorus
    branch: master

- project:
    name: aaa-silicon
    jobs:
      - '{project-name}-verify-{stream}-{maven}-{jdks}'
    stream: silicon
    branch: stable/silicon
```

3. Review and submit the changes to releng/builder project. **(releng/builder committers)**

Autorelease

1. Block submit permissions for registered users and elevate RE's committer rights on Gerrit. **(Helpdesk)**

> **Note:** Enable **Exclusive** checkbox for the submit button to override any existing permissions.

2. Enable create reference permissions on Gerrit for RE's to submit `.gitreview` patches. **(Helpdesk)**

> **Note:** Enable Exclusive checkbox override any existing permissions.

3. Start the branch cut job or use the manual steps below for branch cutting autorelease. **(Release Engineering Team)**
4. Start the version bump job or use the manual steps below for version bump autorelease. **(Release Engineering Team)**
5. Merge all `.gitreview` patches submitted though the job or manually. **(Release Engineering Team)**
6. Remove create reference permissions set on Gerrit for RE's. **(Helpdesk)**
7. Merge all version bump patches in the order of dependencies. **(Release Engineering Team)**
8. Re-enable submit permissions for registered users and disable elevated RE committer rights on Gerrit. **(Helpdesk)**
9. Notify release list on branch cutting work completion. **(Release Engineering Team)**

Branch cut job (Autorelease)

Branch cutting can be performed either through the job or manually.

1. Start the autorelease-branch-cut job **(Release Engineering Team)**

Manual steps to branch cut (Autorelease)

1. Setup `releng/autorelease` repository. **(Release Engineering Team)**

```
git review -s
git submodule foreach 'git review -s'
git checkout master
git submodule foreach 'git checkout master'
git pull --rebase
git submodule foreach 'git pull --rebase'
```

2. Enable create reference permissions on Gerrit for RE's to submit `.gitreview` patches. **(Helpdesk)**

> **Note:** Enable Exclusive check-box override any existing permissions.

3. Create `stable/${CURR_RELEASE}` branches based on HEAD master. **(Release Engineering Team)**

```
git checkout -b stable/${CURR_RELEASE,,} origin/master
git submodule foreach 'git checkout -b stable/${CURR_RELEASE,,} origin/master'
git push gerrit stable/${CURR_RELEASE,,}
git submodule foreach 'git push gerrit stable/${CURR_RELEASE,,}'
```

4. Contribute `.gitreview` updates to `stable/${CURR_RELEASE,,}`. **(Release Engineering Team)**

```
git submodule foreach sed -i -e "s#defaultbranch=master#defaultbranch=stable/${CURR_
↪RELEASE,,}#" .gitreview
git submodule foreach git commit -asm "Update .gitreview to stable/${CURR_RELEASE,,}
↪"
git submodule foreach 'git review -t ${CURR_RELEASE,,}-branch-cut'
sed -i -e "s#defaultbranch=master#defaultbranch=stable/${CURR_RELEASE,,}#" .
↪gitreview
git add .gitreview
git commit -s -v -m "Update .gitreview to stable/${CURR_RELEASE,,}"
git review -t  ${CURR_RELEASE,,}-branch-cut
```

Version bump job (Autorelease)

Version bump can performed either through the job or manually.

1. Start the autorelease-version-bump-${NEXT_RELEASE,,} job **(Release Engineering Team)**

Note: Enabled BRANCH_CUT and disable DRY_RUN to run the job for branch cut work-flow. The version bump job can be run only on the master branch.

Manual steps to version bump (Autorelease)

1. Version bump master by x.(y+1).z. **(Release Engineering Team)**

```
git checkout master
git submodule foreach 'git checkout master'
pip install lftools
lftools version bump ${CURR_RELEASE}
```

2. Make sure the version bump changes does not modify anything under scripts or pom.xml. **(Release Engineering Team)**

```
git checkout pom.xml scripts/
```

3. Push version bump master changes to Gerrit. **(Release Engineering Team)**

```
git submodule foreach 'git commit -asm "Bump versions by x.(y+1).z for next dev␣
↪cycle"'
git submodule foreach 'git review -t ${CURR_RELEASE,,}-branch-cut'
```

4. Merge the patches in order according to the merge-order.log file found in autorelease jobs. **(Release Engineering Team)**

Note: The version bump patches can be merged more quickly by performing a local build with `mvn clean deploy -DskipTests` to prime Nexus with the new version updates.

Documentation post branch tasks

1. Git remove all files/directories from the `docs/release-notes/*` directory. **(Release Engineering Team)**

```
git checkout master
git rm -rf docs/release-notes/<project file and/or folder>
git commit -sm "Reset release notes for next dev cycle"
git review
```

Simultaneous Release

This page explains how the OpenDaylight release process works once the TSC has approved a release.

Code Freeze

At the first Release Candidate (RC) the `Submit` button is disabled on the stable branch to prevent projects from merging non-blocking patches into the release.

1. Disable `Submit` for *Registered Users* and allow permission to the *Release Engineering Team* **(Helpdesk)**

> **Important:** **DO NOT** enable Code-Review+2 and Verified+1 to the Release Engineering Team during code freeze.

> **Note:** Enable **Exclusive** checkbox for the submit button to override any existing permissions. Code-Review and Verify permissions are only needed during version bumping.

This step can be achieved with the self-service job to lock or unlock a Gerrit branch using `autorelease-gerrit-branch-lock-${STREAM}` job on Jenkins CI.

Release Preparations

After release candidate is built GPG sign artifacts using the lftools sign command.

```
STAGING_REPO=autorelease-1903
STAGING_PROFILE_ID=abc123def456  # This Profile ID is listed in Nexus > Staging Profiles
lftools sign deploy-nexus https://nexus.opendaylight.org $STAGING_REPO $STAGING_PROFILE_
↪ID
```

Verify the distribution-karaf file with the signature.

```
gpg2 --verify karaf-x.y.z-${RELEASE}.tar.gz.asc karaf-x.y.z-${RELEASE}.tar.gz
```

Note: Projects such as OpFlex participate in the Simultaneous Release but are not part of the autorelease build. Ping those projects and prep their staging repository as well.

Releasing OpenDaylight

The following describes the Simultaneous Release process for shipping out the binary and source code on release day.

Bulleted actions can be performed in parallel while numbered actions should be done in sequence.

- Release the Nexus Staging repository **(Helpdesk)**
  1. Select both the artifacts and signature repository (*created previously*) and `click Release`.
  2. Enter `Release OpenDaylight $RELEASE` for the description and `click confirm`.

  Perform this step for any additional projects that are participating in the Simultaneous Release but are not part of the autorelease build.

  Tip: This task takes hours to run so kicking it off early is a good idea.

- Version bump for next dev cycle **(Release Engineering Team)**
  1. Run the autorelease-version-bump-${STREAM} job

     Tip: This task takes hours to run so kicking it off early is a good idea.

  2. Enable `Code-Review+2` and `Verify+1` voting permissions for the `Release Engineering Team` **(Helpdesk)**

Note: Enable **Exclusive** checkbox for the submit button to override any existing permissions. Code-Review and Verify permissions are only needed during version bumping. **DO NOT** enable it during code freeze.

  3. Merge all patches generated by the job
  4. Restore Gerrit permissions for *Registered Users* and disable elevated *Release Engineering Team* permissions **(Helpdesk)**

- Tag the release **(Release Engineering Team)**

  1. Install lftools

     lftools contains the version bumping scripts we need to version bump and tag the dev branches. We recommend using a virtualenv for this.

     ```
     # Skip mkvirtualenv if you already have an lftools virtualenv
     mkvirtualenv lftools
     workon lftools
     pip install --upgrade lftools
     ```

  2. Pull latest autorelease repository

     ```
     export RELEASE=Nitrogen-SR1
     export STREAM=${RELEASE//-*}
     export BRANCH=origin/stable/${STREAM,,}

     # No need to clean if you have already done it.
     git clone --recursive https://git.opendaylight.org/gerrit/releng/autorelease
     cd autorelease
     git fetch origin

     # Ensure we are on the right branch. Note that we are wiping out all
     # modifications in the repo so backup unsaved changes before doing this.
     git checkout -f
     git checkout ${BRANCH,,}
     git clean -xdff
     git submodule foreach git checkout -f
     git submodule foreach git clean -xdff
     git submodule update --init

     # Ensure git review is setup
     git review -s
     git submodule foreach 'git review -s'
     ```

  3. Publish release tags

     ```
     export BUILD_NUM=55
     export OPENJDKVER="openjdk11"
     export PATCH_URL="https://s3-logs.opendaylight.org/logs/releng/vex-yul-odl-
     ↪jenkins-1/autorelease-release-${STREAM,,}-mvn35-${OPENJDKVER}/${BUILD_NUM}/
     ↪patches.tar.gz"
     ./scripts/release-tags.sh "${RELEASE}" /tmp/patches "$PATCH_URL"
     ```

- Notify Community and Website teams

  1. Update downloads page

     Submit a patch to the ODL docs project to update the downloads page with the latest binaries and packages **(Release Engineering Team)**

  2. Email dev/release/tsc mailing lists announcing release binaries location **(Release Engineering Team)**

  3. Email dev/release/tsc mailing lists to notify of tagging and version bump completion **(Release Engineering Team)**

Note: This step is performed after Version Bump and Tagging steps are complete.

- Generate Service Release notes

Warning: If this is a major release (eg. Chlorine) as opposed to a Service Release (eg. Chlorine-SR1). Skip this step.

For major releases the notes come from the projects themselves in the docs repository via the *docs/releasenotes/projects* directory.

For service releases (SRs) we need to generate service release notes. This can be performed by running the autorelease-generate-release-notes-$STREAM job.

1. Run the autorelease-generate-release-notes-${STREAM} job **(Release Engineering Team)**

   Trigger this job by leaving a Gerrit comment `generate-release-notes Carbon-SR2`

Release notes can also be manually generated with the script:

```
git checkout stable/${BRANCH,,}
./scripts/release-notes-generator.sh ${RELEASE}
```

A `release-notes.rst` will be generated in the working directory. Submit this file as `release-notes-sr1.rst` (update the `sr` as necessary) to the docs project.

Super Committers

Super committers are a group of TSC-approved individuals within the OpenDaylight community with the power to merge patches on behalf of projects during approved Release Activities.

Super Committer Activities

Super committers are given super committer powers **ONLY** during TSC-approved activities and are not a power that is active on a regular basis. Once one of the TSC-approved activities are triggered, helpdesk will enable the permissions listed for the respective activities for the duration of that activity.

Code Freeze

Note: This activity has been pre-approved by the TSC and does not require a TSC vote. Helpdesk should be notified to enable the permissions and again to disable the permissions once activities are complete or use the self service job to activate the rights for a job.

Super committers are granted powers to merge blocking patches for the duration code of freeze until a release is approved and code freeze is lifted. This permission is only granted for the specific branch that is frozen.

The following powers are granted:

- Submit button access

During this time Super Committers can **ONLY** merge patches that have a +2 Code-Review by a project committer approving the merge, and the patch passes Jenkins Verify check. If neither of these conditions are met then **DO NOT** merge the patch.

Version bumping (Release Work)

Note: This activity has been pre-approved by the TSC and does not require a TSC vote. Helpdesk should be notified to enable the permissions and again to disable the permissions once activities are complete.

Super committers are granted powers to merge version bump related patches for the duration of version bumping activities.

The following powers are granted:

- Vote Code-Review +2
- Vote Verified +1
- Remove Reviewer
- Submit button access

These permissions are granted to allow super committers to push through version bump patches with haste. The Remove Reviewer permission is to be used only for removing Jenkins vote caused by a failed distribution-check job, if that failure is caused by a temporary version inconsistency present while the bump activity is being performed.

Version bumping activities come in 2 forms.

1. Post-release Autorelease version bumping
2. MRI project version bumping

Case 1, the TSC has approved an official OpenDaylight release and after the binaries are released to the world all Autorelease managed projects are version bumped appropriately to the next development release number.

Case 2, During the Release Integrated Deadline of the release schedule MRI projects submit desired version updates. Once approved by the TSC Super Committers can merge these patches across the projects.

Ideally the version bumping activities should not include code modifications, if they do +2 Code-Review vote should be complete by a committer on the project to indicate that they approve the code changes.

Once version bump patches are merged these permissions are removed.

Exceptional cases

Any activities not in the list above will fall under the exceptional case in which requires TSC approval before Super Committers can merge changes. These cases should be brought up to the TSC for voting.

Self-service

All of the above states (super committers, code freeze, release work, version bump) on the Gerrit branch can be achieved using a self-serviced Jenkins CI job `autorelease-gerrit-branch-lock-${STREAM}`. The job be triggered by anyone in the releng committers group.

Note: Before starting the job to lock/unlock the branch, check the branch state using the link.

https://git.opendaylight.org/gerrit/gitweb?p=All-Projects.git;a=blob_plain;f=project.config;hb=HEAD

Super Committers

Name	IRC	Email
Anil Belur	abelur	abelur@linux.com
Jamo Luhrsen	jamoluhrsen	jluhrsen@gmail.com
Luis Gomez	LuisGomez	ecelgp@gmail.com
Robert Varga	rovarga	nite@hq.sk
Thanh Ha	zxiiro	zxiiro@gmail.com

Supporting Documentation

Identifying Managed Projects in an OpenDaylight Version

What are Managed Projects?

Managed Projects are simply projects that take part in the *Managed Release Process*. Managed Projects are either core components of OpenDaylight or have demonstrated their maturity and ability to successfully take part in the Managed Release.

For more information, see the full description of *Managed Projects*.

What is a Managed Distribution?

Managed Projects are aggregated together by a POM file that defines a Managed Distribution. The Managed Distribution is the focus of OpenDaylight development. It's continuously built, tested, packaged and released into Continuous Delivery pipelines. As prescribed by the Managed Release Process, Managed Distributions are eventually blessed as formal OpenDaylight releases.

NB: OpenDaylight's Fluorine release actually included Managed and Self-Managed Projects, but the community is working towards the formal release being exactly the Managed Distribution, with an option for Self-Managed Projects to release independently on top of the Managed Distribution later.

Finding the Managed Projects given a Managed Distribution

Given a Managed Distribution (`.tar.gz`, `.zip`, RPM, Deb), the Managed Projects that constitute it can be found in the *taglist.log* file in the root of the archive.

taglist.log files are of the format:

```
<Managed Project> <Git SHA of built commit> <Codename of release>
```

Finding the Managed Projects Given a Branch

To find the current set of Managed Projects in a given OpenDaylight branch, examine the integration/distribution/features/repos/index/pom.xml file that defines the Managed Distribution.

The release management team maintains several documents in Google Drive to track releases. These documents can be found at the following link:

https://drive.google.com/drive/folders/0ByPlysxjHHJaUXdfRkJqRGo4aDg

3.2.3 Java API Documentation

Release Integrated Projects

- odlparent
- yangtools
- mdsal

Managed Projects

- bgpcep
- genius
- infrautils
- lispflowmapping
- netvirt
- neutron
- openflowplugin
- ovsdb

3.2.4 OpenDaylight User Guide

Overview

This first part of the user guide covers the basic user operations of the OpenDaylight Release using the generic base functionality.

OpenDaylight Controller Overview

The OpenDaylight controller is JVM software and can be run from any operating system and hardware as long as it supports Java. The controller is an implementation of the Software Defined Network (SDN) concept and makes use of the following tools:

- **Maven**: OpenDaylight uses Maven for easier build automation. Maven uses pom.xml (Project Object Model) to script the dependencies between bundle and also to describe what bundles to load and start.
- **OSGi**: This framework is the back-end of OpenDaylight as it allows dynamically loading bundles and packages JAR files, and binding bundles together for exchanging information.
- **JAVA interfaces**: Java interfaces are used for event listening, specifications, and forming patterns. This is the main way in which specific bundles implement call-back functions for events and also to indicate awareness of specific state.
- **REST APIs**: These are northbound APIs such as topology manager, host tracker, flow programmer, static routing, and so on.

The controller exposes open northbound APIs which are used by applications. The OSGi framework and bidirectional REST are supported for the northbound APIs. The OSGi framework is used for applications that run in the same address space as the controller while the REST (web-based) API is used for applications that do not run in the same address space (or even the same system) as the controller. The business logic and algorithms reside in the applications. These applications use the controller to gather network intelligence, run its algorithm to do analytics, and then orchestrate the new rules throughout the network. On the southbound, multiple protocols are supported as plugins, e.g. OpenFlow 1.0, OpenFlow 1.3, BGP-LS, and so on. The OpenDaylight controller starts with an OpenFlow 1.0 southbound plugin. Other OpenDaylight contributors begin adding to the controller code. These modules are linked dynamically into a **Service Abstraction Layer** (SAL).

The SAL exposes services to which the modules north of it are written. The SAL figures out how to fulfill the requested service irrespective of the underlying protocol used between the controller and the network devices. This provides investment protection to the applications as OpenFlow and other protocols evolve over time. For the controller to control devices in its domain, it needs to know about the devices, their capabilities, reachability, and so on. This information is stored and managed by the **Topology Manager**. The other components like ARP handler, Host Tracker, Device Manager, and Switch Manager help in generating the topology database for the Topology Manager.

For a more detailed overview of the OpenDaylight controller, see the *OpenDaylight Developer Guide*.

Project-specific User Guides

- JSON RPC Documentation
- OVSDB Documentation

Distribution Version reporting

Overview

This section provides an overview of **odl-distribution-version** feature.

A remote user of OpenDaylight usually has access to RESTCONF and NETCONF northbound interfaces, but does not have access to the system OpenDaylight is running on. OpenDaylight has released multiple versions including Service Releases, and there are incompatible changes between them. In order to know which YANG modules to use, which bugs to expect and which workarounds to apply, such user would need to know the exact version of at least one OpenDaylight component.

There are indirect ways to deduce such version, but the direct way is enabled by `odl-distribution-version` feature. Administrator can specify version strings, which would be available to users via NETCONF, or via RESTCONF if OpenDaylight is configured to initiate NETCONF connection to its config subsystem northbound interface.

By default, users have write access to config subsystem, so they can add, modify or delete any version strings present there. Admins can only influence whether the feature is installed, and initial values.

Config subsystem is local only, not cluster aware, so each member reports versions independently. This is suitable for heterogeneous clusters.

Default config file

Initial version values are set via config file `odl-version.xml` which is created in `$KARAF_HOME/etc/opendaylight/karaf/` upon installation of `odl-distribution-version` feature. If admin wants to use different content, the file with desired content has to be created there before feature installation happens.

By default, the config file defines two config modules, named `odl-distribution-version` and `odl-odlparent-version`.

RESTCONF usage

OpenDaylight config subsystem NETCONF northbound is not made available just by installing `odl-distribution-version`, but most other feature installations would enable it. RESTCONF interfaces are enabled by installing `odl-restconf` feature, but that do not allow access to config subsystem by itself.

On single node deployments, installation of `odl-netconf-connector-ssh` is recommended, which would configure `controller-config` device and its MD-SAL mount point.

For cluster deployments, installing `odl-netconf-clustered-topology` is recommended. See documentation for clustering on how to create similar devices for each member, as `controller-config` name is not unique in that context.

Assuming single node deployment and user located on the same system, here is an example `curl` command accessing `odl-odlparent-version` config module:

```
curl 127.0.0.1:8181/restconf/config/network-topology:network-topology/topology/topology-
↪netconf/node/controller-config/yang-ext:mount/config:modules/module/odl-distribution-
↪version:odl-version/odl-odlparent-version
```

Neutron Service User Guide

Overview

This Karaf feature (`odl-neutron-service`) provides integration support for OpenStack Neutron via the OpenDaylight ML2 mechanism driver. The Neutron Service is only one of the components necessary for OpenStack integration. For those related components please refer to documentations of each component:

- https://wiki.openstack.org/wiki/Neutron
- https://launchpad.net/networking-odl
- https://opendev.org/openstack/networking-odl/
- https://wiki-archive.opendaylight.org/view/NeutronNorthbound:Main

Use cases and who will use the feature

If you want OpenStack integration with OpenDaylight, you will need this feature with an OpenDaylight provider feature like netvirt, group based policy, VTN, and lisp mapper. For provider configuration, please refer to each individual provider's documentation. Since the Neutron service only provides the northbound API for the OpenStack Neutron ML2 mechanism driver. Without those provider features, the Neutron service itself is not useful.

Neutron Service feature Architecture

The Neutron service provides northbound API for OpenStack Neutron via RESTCONF and also its dedicated REST API. It communicates through its YANG model with providers.

Configuring Neutron Service feature

As the Karaf feature includes everything necessary for communicating northbound, no special configuration is needed. Usually this feature is used with an OpenDaylight southbound plugin that implements actual network virtualization functionality and OpenStack Neutron. The user wants to setup those configurations. Refer to each related documentations for each configurations.

Administering or Managing `odl-neutron-service`

There is no specific configuration regarding to Neutron service itself. For related configuration, please refer to OpenStack Neutron configuration and OpenDaylight related services which are providers for OpenStack.

installing `odl-neutron-service` while the controller running

1. While OpenDaylight is running, in Karaf prompt, type: `feature:install odl-neutron-service`.
2. Wait a while until the initialization is done and the controller stabilizes.

`odl-neutron-service` provides only a unified interface for OpenStack Neutron. It does not provide actual functionality for network virtualization. Refer to each OpenDaylight project documentation for actual configuration with OpenStack Neutron.

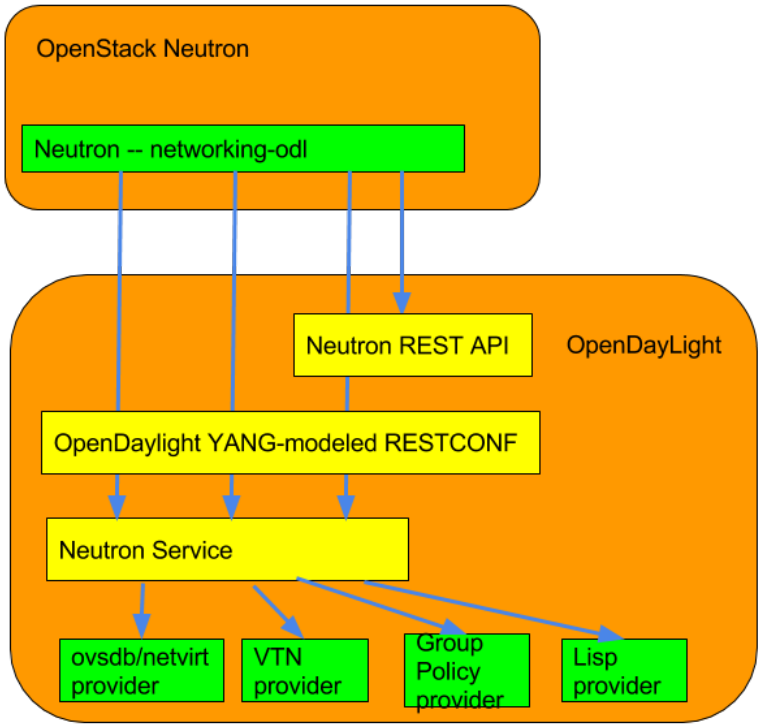

Fig. 1: Neutron Service Architecture

Neutron Logger

Another service, the Neutron Logger, is provided for debugging/logging purposes. It logs changes on Neutron YANG models.

```
feature:install odl-neutron-logger
```

NETCONF User Guide

Guide has moved. Please navigate to document from here

3.2.5 Contributor Guides

Newcomers Guide

What is Gerrit ?

The Linux Foundation proposes a LF Gerrit Guide at this URL: https://docs.releng.linuxfoundation.org/en/latest/gerrit.html Please look at it.

You can find more details on Gerrit by itself at https://gerrit-review.googlesource.com/Documentation/intro-quick.html

Gerrit is a web-based code review tool built on top of the Git version control system. Gerrit makes code review easier by providing a lightweight framework for reviewing commits before they are accepted by the codebase i.e. project committers.

If you are not familiar with Git, please look at the LFN Git guide.

Gerrit uses massively git hooks. Git hooks are commands and scripts to add automatically specific treatments in git basic commands. **The main hook treatment provided by Gerrit is to add a "Change-Id" tag to all commit messages. This "Change-Id" allows Gerrit to link together different versions of the same change being reviewed.**

If you want to look at the hooks contents, you can retrieve them manually on the Gerrit git server with those commands.

```
$ scp -p -P 29418 <login>@git.opendaylight.org:hooks/commit-msg RecipeBook/.git/hooks/
$ chmod u+x .git/hooks/commit-msg
```

Most of the time, retrieving hooks manually is useless since they are automatically invoked by git commands. With Gerrit, it is also much easier to use git-review that invokes those hooks. **Git-review is a tool that helps submitting git branches to Gerrit for review, this means it can supersede the usual git fetch, git pull and git push commands.** The git-review package is available on most GNU/Linux distribution.

In Gerrit, several ongoing-review branches or "changes" can coexist without conflicts until they are merged to a mainstream branch (usually the master). This way, several paths for the project can be explored simultaneously even if they implement a same feature. However, if a change is getting too old, it may become impossible to merge because the mainstream branch has evolved too much meanwhile. In that case, you have to rebase the change on the mainstream branch and resolve potential conflicts before it can be merged.

Guidelines and practical advice

clone a project repository and get a local copy of the code

To access OpenDaylight repository, you need a Linux Foundation ID to log in. This can be done at this URL.

Once that done, you need to generate your SSH keys pair and publish your public on your ODL Gerrit account as described in this link.

To clone the current master branch of a project

```
$ git clone ssh://<login>@git.opendaylight.org:29418/<project_name>
```

To clone another mainstream branch

```
$ git clone -b <branch_name> ssh://< login >@git.opendaylight.org:29418/<project_name>
```

for example the magnesium branch of the documentation project

```
$ git clone -b stable/magnesium ssh://my_login@git.opendaylight.org:29418/docs
```

at that step, it is usually a good idea to run the setup commands of the repository with git-review

```
$ cd docs/
$ git-review -s
```

If needed, a particular change (i.e. a specific branch used for staging reviews before they are merged to the master branch or another mainstream branch) can be retrieved by using git-review

```
$ git-review -d <change_number>
```

This will create another local repository along the master branch copy. You can use git branch to verify it.

prepare a change

Once you have a local copy of the repository, you can make your modifications. Please follow the best practices given below and in *Coding Guidelines*. Remember to check what you have done.

Be particularly careful to the license headers, the trailing blanks, the empty lines and do not use the MSDOS file format but the UNIX file format. Try also to remove compilation warnings.

If you are using an IDE , it can be a good idea to use an editor profile that implements those rule such as the Eclipse profile in this link.

Since ODL compilation process is particularly verbose, consider using compilation logs file or piped commands such as:

```
$ time mvn clean install -DskipTests 2>&1 |tee mvn.log.0 |grep 'WARN\|ERROR'
```

If you propose an update for an existing feature with tests already available, it is a good idea to perform these tests (e.g. "$ tox -e pre-commit") and see what happens. Available tests can often be found in the `tox.ini` file at the root of the project folder.

Once ready, use git status to check the staging files.

```
$ git status
```

If you want to commit all your changes, you can skip the next 2 steps and use directly "$ git commit" with the option "-a"

If not, add the right files to your commit.

```
$ git add [ ... ]
```

Note: You might want to remove some files from the remote repository in this commit. In that case, you should use `git rm` instead of only `rm`.

The same way, use `git mv` if you want to rename or move a file in the remote repository too.

It is a good idea to check again your git status before committing.

```
$ git status
```

Then commit and add a commit message. Using "-s" to sign-off your commit is usually a good idea.

```
$ git commit -s
```

Note: Please abide by the commit messages rules given below and at the openStack wiki.

Be careful with the title length (50 char), the empty line after the title, and the body length (72 char).

If your commit includes work from other contributors, do not hesitate to use the "co-authored-by" tag.

If you are not the author of the changes, you can upload it although but you should use the option "–author=" with "git-commit"

Contributors must agree with the OpenDaylight Technical charter. The sign-off field is related to the role of the Certificate of Origin explained in this charter.

At that step, you can still rework your modifications and include more files with "git add". In that case, amend the commit after with the command.

```
$ git commit --amend
```

This command also allows you to rework your commit message too.

Upload a Change

Uploading a change to Gerrit is done by pushing a git commit to the Gerrit origin server. The commit must be pushed to a ref in the refs/for/ namespace which defines the target branch: refs/for/< target-branch >. The magic refs/for/ prefix allows Gerrit to differentiate commits that are pushed for review from commits that are pushed directly into the repository, bypassing code review (this is usually a bad idea). For the target branch it is sufficient to specify the short name, e.g. master, but you can also specify the fully qualified branch name, e.g. refs/heads/master.

Push for Code Review

```
$ git commit
$ git push origin HEAD:refs/for/master

// this is the same as:
```

(continues on next page)

(continued from previous page)

```
$ git commit
$ git push origin HEAD:refs/for/refs/heads/master
```

It is easier to use the equivalent git-review commands. The -T option allows to avoid sending the local branch name as default topic.

```
$ git-review -T
```

If you want to upload it on another mainstream branch for review, you can add the branch name at the end.

```
$ git-review -T <branch_name>
```

for example magnesium

```
$ git-review -T stable/magnesium
```

It is also sometimes possible to push commits with bypassing Code Review. Beware this is usually a bad idea !

```
$ git commit
$ git push origin HEAD:master

// this is the same as:
$ git commit
$ git push origin HEAD:refs/heads/master
```

Check your change on Gerrit

Each file added, modified, moved, renamed or deleted will be listed in the Gerrit page displaying your change. If you click on a file name, the differences with the previous version of the file will be displayed. Main common errors such as trailing blanks usually appears in red. Please check every file and list those common errors. Then (or in parallel) you can go to the next steps and correct them quickly. This is a good idea to do it before asking other people to review your change in Gerrit.

Upload a new Patch Set

If there is feedback from code review and a change should be improved, a new patch set with the reworked code should be uploaded.

This is done by amending the commit of the last patch set.

If you have no more a local copy of your change, you can use "git clone" and "git-review -d" to retrieve it (just as described in the first section " The commit can also be fetched from Gerrit by using the fetch commands available from the download commands in the change screen (right top corner).

```
// fetch and checkout the change
// (checkout command can be copied from change screen, right top corner in download)
$ git fetch "https://git.opendaylight.org/gerrit/docs" refs/changes/86/93386/2
  && git checkout FETCH_HEAD

// or with git-review
$ git-review -d 93386
```

(continues on next page)

(continued from previous page)

```
// provided 2 is the latest Patch Set, otherwise if there is a Patch Set 3,
$ git-review -d 93386,2
// specifying a Patch-Set number only works with newer git-review versions
```

Then you can start working on it just as you will do for a new commit with `git add/rm/mv` etc. Once ready, instead of simply doing `$ git commit -s` type instead `$ git commit --amend`

```
// rework the change
$ git add < path-of-reworked-file > [ ... ]

// amend commit
$ git commit --amend

// push patch set
$ git push origin HEAD:refs/for/master
// or with git-review
$ git-review -T
```

It is important that the commit message contains the Change-Id of the change that should be updated as a footer (last paragraph). Normally the commit message already contains the correct Change-Id and the Change-Id is preserved when the commit is amended.

Thanks to the Change-Id in the commit message, Gerrit will detect that the change was already there and that you want to create a new Patch Set to amend it. The new Patch Set should now appear in the Gerrit web interface.

Note: Never amend a commit that is already part of a central branch.

Pushing a new patch set will trigger an email notification to the reviewers who subscribed to the project notifications.

The option -T is used to avoid adding a topic to the change. If no topic is specified, git-review will add the change number or the local branch name as a topic in Gerrit web interface. You can force another topic with the -t option.

Submitting simultaneously several changes for review

Sometimes, it can be interesting to push simultaneously several interdependent changes for review. This can be done this way.

```
$ git add [...]

$ git commit -s

[ ... ]

$ git add [...]

$ git commit -s

$ git-review
```

Here is a simple example that modifies an existing change and proposes a new change on top of it.

```
$ git clone https://git.opendaylight.org/gerrit/docs
$ git-review -s
$ git-review -d 93386

// rework the change 93386
[..]
$ git add < path-of-reworked-file > [ ... ]

// amend commit
$ git commit -amend

// add a new change/commit
$ git add < path-of-worked-file > [ ... ]
// add a new commit
$ git commit

// repeat the operation as much time as necessary
[..]

// upload the changes to Gerrit
$ git-review -T
```

git-review usually displays a warning and ask confirmation when doing this. The option -y avoids this message.

If the changes are accepted, the Gerrit web interface will display information a.k.a. relation chain on changes submitted together when looking at one of them.

Note: When cascading more changes, the first call "git-review" may fail because of the absence of a Change-Id in the git commit message logs.

Retry "git-review" in that case or try to run git hook manually to modify the git log history (not so easy).

If you do not have Gerrit git hooks pre-installed, this only works for the absence of Change-Id in the last commit. In that case, you can use interactive rebase with reword to edit the N previous commit messages ("git-rebase -i HEAD~N").

Modify several changes

While they have not been merged in the remote repository, it is still possible to rework the changes that were posted simultaneously. If you have no more a local copy of them, just retrieve the latest change in you git history from the Gerrit remote repository. Check the history with

```
$ git log
```

It should display all the commits posted.

"git commit –amend" only allows to rework the last commit. You must use another method to rework the previous commits.

The easiest way to do that is to use interactive rebase 2 syntaxes can be used:

```
$ git rebase -i < commit >
```

where is the commit hash reference used by "git log"

or

```
$ git rebase -i HEAD~< number of commits >
// e.g. to rework the five previous commits
$ git rebase-i HEAD~5
```

you should now see commits short descriptions in a text editor (usually vim) It should look like this.

```
pick 239da71 Renderer and OLM update
pick f85398e Bugs correction in Portmapping
pick 6cb0144 Minor checkstyle corrections
pick e51e0b9 Network topology and inventory init
pick f245366 Bugs correction in NetworkModelService

# Rebase afe9fcf..f245366 onto afe9fcf
#
# Commands:
#  p, pick = use commit
#  r, reword = use commit, but edit the commit message
#  e, edit = use commit, but stop for amending
#  s, squash = use commit, but meld into previous commit
#  f, fixup = like "squash", but discard this commit's log message
#  x, exec = run command (the rest of the line) using shell
#
# These lines can be re-ordered; they are executed from top to bottom.
#
# If you remove a line here THAT COMMIT WILL BE LOST.
#
# However, if you remove everything, the rebase will be aborted.
#
# Note that empty commits are commented out
```

The editor allows you to proceed to several actions on the git history:

- remove a commit from the history: just delete its line
- rework dependencies: swap line orders
- meld several commits into one: replace `pick` by `squash` or `fixup`
- rework only a specific commit message: replace `pick` by `reword`
- rework a specific commit: replace `pick` by `edit` then `git add/rm/mv ...`, `git commit --amend`, `git rebase --continue`

Beware you may have to deal with potential conflicts when doing this.

Note that alternate methods exist. For example, you can use cherry-picks described in the next section. You can also use non-interactive git rebase , i.e without the option "-i". But you must keep a copy of the original "git log" history. Most people create a new local branch with a copy via "git checkout -b" to that purpose. Once the copy made, use

```
$ git checkout <commit_hash>
```

where < commit_hash > is the hash of a previous commit, let' say N commits before the last one. Do your modifications:

```
$ git add/rm/mv [...]
$ git commit --amend
```

A new commit hash (<newhash>) will be generated. Keep it.

```
$ git checkout <commit_hash-1>
```

where < commit_hash-1 > is the hash of the previous commit, N-1 commits before the last one. If you look at "git log", the history has not changed and the old hash is still there. you need to rebase to apply the modifications made in the previous commit.

```
$ git log
$ git rebase <newhash>
$ git log
```

Conflicts may appear but should be solvable. Proceed the same way with the N-2 previous commits up to the last commit. Then upload

```
$ git-review
```

Cherry-picks / back-ports

Cherry-pick consist in importing the content of a specific change (or commit) from another (review) branch into your own local branch.

The basic git cherry-pick method is described in the LFN Git Guide.

The principle remains the same with Gerrit but you have to deal with the Gerrit branch review system. You can use the "git cherry-pick" classical command. In that case, you'd better to copy/paste it from the right-top corner of the change review page. The easiest option is to use git-review with the option "-x" instead.

```
$ git-review -x < change_number >"
```

You can use also "-X" to keep a trace of the cherry-pick operation in the git log. The "-N" option prepare the cherry-pick but the commit message is not imported.

Several cherry-picks can be cascaded this way.

Once the change appears in your local branch, you can upload it to the Gerrit remote repository as usual with git-review.

Cherry-pick can also be used to back-port changes between several mainstream branches of the Gerrit remote repository. The procedure is the same. Here is an example.

```
$ git clone -b stable/aluminium ssh://< login >@git.opendaylight.org:29418/docs
$ git-review -x 94257
//Change 94257 is on the magnesium branch and not the aluminium branch
$ git-review [-P] [stable/aluminium]
```

Resolving conflicts

Conflict resolution in Gerrit is not different from Git. You can also refer to the LFN Git Guide.

Conflict can occur during Git merge, push or rebase operations.

For example, if two or more members make changes on the same part of a file in a remote and a local branch that are being merged, Git will not be able to automatically merge them and you will get a merge conflict. When this happens, conflicting files will be listed in the resulting messages as in the example below.

```
$ git merge issue3
Auto-merging my_shopping_list.txt CONFLICT (content): Merge conflict in my_shopping_list.
↪txt
Automatic merge failed; fix conflicts and then commit the result.
```

And Git will add some standard conflict-resolution markers to those conflicting files. The markers act as an indicator to help us figure out sections in the content of the conflicting file that needs to be manually resolved.

Example of a conflict occurrence

```
My Shopping list

Apples
<<<<<<< HEAD
Bread
Pancakes
=======
Banana
Soda
>>>>>>> issue3
Tomatoes
```

Each conflicting section in the file is delimited by lines alike "<<<<<<< HEAD" and ">>>>>>> issue3" . When merging remote code into your local branch, everything above " ======== " is your local content, and everything below it comes from the remote branch. Before going further, we need to resolve the conflicting parts and removes those markers as shown in the example below.

```
My Shopping list

Apples
Banana
Bread
Pancakes
Soda
Tomatoes
```

Once we are done with resolving the conflict, you can commit the change (git commit -m) , or pursue a rebase if you were in a rebasing process.

OpenDaylight and common Best Practices

All details on OpenDaylight best practices are available at this URL on the old wiki.

Implicit rules

The first rule is that the author or at least the owner(=uploader) of the change is responsible for the code posted on the Gerrit server. This means that the author or the owner has to be responsive to questions in Gerrit comments. This is especially important for adaptations asked by committers. Committers are in charge of making the mainstream branch clean and conform to the project rules before merging it in the mainstream branch. Other reviews from non-committers are also welcome.

It may sound a little awkward but **many developers consider a "-1" review as good news as a "+1" review .Both mean someone has looked at their code and posted useful comments, potentially reusable elsewhere.**

There can be several interpretations of what to do in some case and Gerrit comments can be very useful to clarify points in case of disagreements. If possible, the change uploader or owner must be the code author so that the review is more interactive and responsive.

The second rule is to keep the code posted reviewable. The change should not bring regression nor new compilation errors and warnings. It is a good idea to look at the Gerrit interface editor once your code has been posted for review. Most common errors such as trailing blanks are colored in red. Those errors pollutes the review process, not only because they generate many warnings during the compilation process. Posting a quick fix for those most common issues in a new Patch Set will ease the reviewers and committers work. If you are not confident of what you have done, you can test your change in Gerrit by using the private before making it public or by using the Work-in-Progress mode to clearly state it is an ongoing work.

Huge amounts of code are also generally difficult to review. Gerrit changes dashboard has a size indicator on the right. There is no strict rule about this but if you receive a XL size, you probably should consider to split your change into several smaller ones.

Coding Guidelines and common issues

More details at *Coding Guidelines*.

Commit message

More details at https://wiki.openstack.org/wiki/GitCommitMessages

The commit message should reflect the feature or improvements posted in the change. The message must give a good idea of what have been done. **It must be understood by anybody with a sufficient knowledge on the topic, not necessarily someone taking part to the project.** External references are welcome to point out to more details, but they should not be a substitute to a correct description.

Here is a summary of Git commit message structure as described in the OpenStack wiki.

- Provide a brief description of the change in the first line.
- Insert a single blank line after the first line.
- Provide a detailed description of the change in the following lines, breaking paragraphs where needed.
- The first line should be limited to 50 characters and should not end with a period.

- Subsequent lines should be wrapped at 72 characters. There are some exceptions to this rule: for example, URL should not be wrapped. Vim (the default $EDITOR on most distributions) can wrap automatically lines for you. In most cases you just need to copy the example `vimrc` file (which can be found somewhere like */usr/share/vim/vim74/vimrc_example.vim* `) to `~/.vimrc` if you don't have one already. After editing a paragraph, you can re-wrap it by pressing escape, ensuring the cursor is within the paragraph and typing `gqip`. Put the 'Change-id', 'Closes-Bug #NNNNN' and 'blueprint NNNNNNNNNNN' lines at the very end.

Note: It is common practice across many open source projects using Git to include a one or several "Signed-off-by" tags (generated by 'git commit -s'). If the change has several authors, you are encouraged to use the 'Co-authored-by' tag. Relate tickets, tasks and bug issues are pointed in the commit message using the JIRA tag.

Files formatting

Files posted for review should use the UNIX/linux file format. This means that their line terminator is "\n" aka LF or LineFeed. **Other format such as MSDOS (with "\rn" aka CRLF aka Carriage Return Line Feed terminators) should be avoided.** Encoding formats commonly accepted are Unicode and ASCII.

You can use the `file` GNU command to check the actual status of your files.

```
$ file *
activate-projects-rtd-branch.sh: Bourne-Again shell script, ASCII text executable
branch-cutting-checklist.txt:    ASCII text
ci-requirements.txt:             ASCII text
docs:                            directory
find_bad_words.sh:               ASCII text
INFO.yaml:                       ASCII text
README.md:                       ASCII text
tox.ini:                         ASCII text
web:                             directory
```

and combine it with `find` and `xargs` + `grep` to detect MSDOS file

```
$ find . | xargs file | grep CRLF
 ./tox.ini:  ASCII text, with CRLF line terminators
 ./docs/make.bat:  DOS batch file, ASCII text, with CRLF line terminators
```

then create a script with `sed` to remove the "\r" special character and convert them in the UNIX format.

```
$ find . | xargs file | grep CRLF  |grep -v make.bat | cut -d':' -f1 | xargs sed -i 's/\\
↪r//'
```

More easily, the vim editor can convert MSDOS file to UNIX format with `:set ff=unix` If you are on windows, do not use notepad since it only uses the MSDOS format. Consider using `textpad++` or another advanced editor.

The ODL Java style guide limits the Java files line length to 120 characters and 72 or 80 chars for javadoc. It prohibits also the use of tabs. They should be replaced with 4 white-spaces. Below is a shell script to automate the operation inside a folder.

```
$ for file in * ; do sed -i 's/\\t/ /g' $file; done
```

Trailing blanks should be avoided too. Below is a shell script to remove trailing white-spaces inside a folder.

```
$ for file in * ; do sed -i 's/ \*//' $file;done
```

Useless empty lines must also be avoided.

Note: If you are using an operating system with a different default version of `sed` than GNU `sed`, for example BSD `sed` on MAC OS X, you must adapt the examples given here since the -i option takes a mandatory parameter.

License issues

The EPL license or a compatible license should be present on all projects code file in the header. The maven java Checkstyle plugin will check the presence of this license.

More details in the *Coding Guidelines*.

License issues are considered particularly sensible by the open-source communities.

Coding Guidelines

Generic Coding Guidelines

Note: This document is a work in progress.

Git commit message style

OpenDaylight projects store code in git repositories, so contributions to OpenDaylight code have form of git commits. Each git commit needs a proper commit message.

The message should describe what has changed from a previous state. Ideally this will be the preceding commit in git history. OpenDaylight uses Gerrit to manage reviewing and merging of contributions, so in this context "previous state" is shown as Base.

Note: Chris Beams'blog is pretty descriptive of why formatting the commit message properly is useful.

For Git commit messages we recommend following the OpenStack commit message recommendations.

- Separate subject from body with a blank line
- Limit the subject line to 50 characters
- Capitalize the subject line
- Do not end the subject line with a period
- Use the imperative mood in the subject line
- Wrap the body at 72 characters
- Use the body to explain what and why vs. how

During review, reviewers will comment on current patch sets, and contributor may add more edits into newer patch sets. But previous patch sets are NOT the previous git state the commit message should document against. Contributors should respond to reviewers via Gerrit posts, while updating commit message to track current differences from Base.

The commit message is most frequently read by people examining "git log" output, and they usually do not have information on which files were edited. Make sure your commit message is specific enough. For example, write "Add option X to module Y" instead of just "Add option X".

As there are tighter limits on the first line of commit message, the end of the line should not be marked by period. sentences in the rest of commit message should be punctuated as usual.

Some projects verify commit message formatting in automated verify-style job. For example Releng/Builders enforces most of Coala rules.

General Code headers

License and Copyright headers need to exist at the top of all code files. Examples of copyright headers for each language can be seen below.

Note: In case you need multiple Copyright headers simply duplicate the Copyright line for additional copyrights

C/C++/Java

```
/*
 * Copyright (c) 2021 <Company or Individual>.  All rights reserved.
 *
 * This program and the accompanying materials are made available under the
 * terms of the Eclipse Public License v1.0 which accompanies this distribution,
 * and is available at http://www.eclipse.org/legal/epl-v10.html
 */
```

Bash/Python

```
##############################################################################
# Copyright (c) 2021 <Company or Individual>.  All rights reserved.
#
# This program and the accompanying materials are made available under the
# terms of the Eclipse Public License v1.0 which accompanies this distribution,
# and is available at http://www.eclipse.org/legal/epl-v10.html
##############################################################################
```

XML

```
<!--
  Copyright (c) 2021 <Company or Individual>.  All rights reserved.

  This program and the accompanying materials are made available under the
  terms of the Eclipse Public License v1.0 which accompanies this distribution,
  and is available at http://www.eclipse.org/legal/epl-v10.html
-->
```

General Code Style

- 120 characters line length

Coding Guidelines for Java

Note: This document is a work in progress.

In General we follow the Google Java Code Style Guide with a few exceptions. See: https://google.github.io/styleguide/javaguide.html

- 4 spaces indentation
- 120 characters line length
- 72 or 80 characters for Javadoc
- import ordering (https://wiki-archive.opendaylight.org/view/GettingStarted:_Eclipse_Setup#Import_ordering)
- YangTools Design Guidelines (https://wiki-archive.opendaylight.org/view/YANG_Tools:Design_and_Coding_Guidelines)
- follow JLS modifier ordering (https://checkstyle.sourceforge.io/checks/modifier/modifierorder.html)
- do not use underscores (_) in identifiers, as they trigger warnings with JDK8
- Uppercase " *static final Logger LOG* = ... " (See 5.2.4 examples)

Checkstyle

All OpenDaylight projects automatically run Checkstyle, as the `maven-checkstyle-plugin` is declared in the odlparent.

Checkstyle Warnings are considered as errors by default since Magnesium. They will prevent your project from building if code violations are found. Checkstyle enforcement can be disabled by defining the following property in the related pom.xml:

```
<properties>
 <odlparent.checkstyle.enforce>false</odlparent.checkstyle.enforce>
</properties>
```

Utility classes with only static methods

In general we recommend that you typically do not overuse utility classes with only static methods, but instead write helper @Singleton classes without static methods which you can easily @Inject via Dependency Injection into other classes requiring them. This makes it easier to use non-static helpers in other utilities which can then in turn be @Inject into your helper (which you cannot do with static). It also makes it easier to mock the helpers for use in unit tests.

If you must write utility classes with only static methods, or have existing code that is not trivial to change, then please mark the respective class final, and give it a private constructor. Please do not throw any exception from the private constructor (it is not required).

Suggested process (steps) to move a non-compliant project to enforcement

We recommend moving existing at least large projects (which typically have hundreds or thousands of Checkstyle violations) to full compliance and enforcement through a series of Gerrit changes on single artefacts (bundles), as opposed to a single change fixing everything and change the POM to enable enforcement all in one go (god forbid for an entire repository and not just a single artifact), because:

1. single review would be virtually impossible to even remotely sensibly code review by committers
2. batching style changes by type is easy to review (and approve lines in bulk "by trust"), for example:
    1. *(...project name...) Organize Imports for Checkstyle compliance*
    2. *(...project name...) Checkstyle compliance: line length*
    3. *(...project name...) Checkstyle compliance: various types of small changes*
    4. ''(...project name...) Checkstyle compliant Exception handling'
    5. ''(...project name...) Checkstyle final clean up & enforcement'
3. it's particularly important to split and separately submit "trivial pure cosmetic" (e.g. code formatting) from "interesting impactful" (e.g. changes to exception handling) changes required by Checkstyle
4. in general, doing small steps and intermediate merges are more rewarding for contributing developers than long running massive Gerrit changes
5. more small changes makes the contributors Stats Great Again (ODL top contributors submit massive amounts of micro changes)

During such a process, it should be considered "normal" and perfectly acceptable, that new intermediately merged changes introduce some (small) regressions and "re-dirty" some previously cleaned up files; it's OK that they'll be re-fixed as part of new changes by the developers contributing the clean up in new changes (or rebases) - until enforcement is enabled at the end of a series of clean up changes.

`@SuppressWarnings`

If really needed, projects can override individual Checkstyle rules on a case-by-case basis by using a `@SuppressWarnings` annotation:

```
@SuppressWarnings("checkstyle:methodparampad")
public AbstractDataTreeListener (INetvirtSfcOF13Provider provider, Class<T>␣
↪clazz) {
}
```

The rule ID (e.g. `checkstyle:methodparampad` above) is the name of the respective Checkstyle module; these IDs can be found e.g. in the `git/odlparent/checkstyle/src/main/resources/odl_checks.xml` configuration, or directly on the Checkstyle website from the http://checkstyle.sourceforge.net/checks.html list. For example, for the http://checkstyle.sourceforge.net/config_coding.html#EqualsHashCode rule you would put `@SuppressWarnings("checkstyle:EqualsHashCode")`.

This `@SuppressWarnings("checkstyle:...")` should in practice be very very rarely needed. Please put a comment explaining why you need to suppress a Checkstyle warning into the code for other to understand, for example:

```
@Override
@SuppressWarnings("checkstyle:EqualsHashCode"
// In this particular case an equals without hashCode is OK because
// [explain!] (I'm a certified grown up and know what I'm doing.)
public boolean equals(Object obj) {
```

Please contact odlparent-dev@lists.opendaylight.org if you feel a Checkstyle rule is too strict in general and should be reviewed.

The Evolving Checkstyle old wiki page documents how to test changes to Checkstyle rules.

Notes for particular Checks

{@inheritDoc} JavaDoc

This JavaDoc is useless visual noise that hinders code readability. It is not required to put this, and has no impact. JavaDoc does this by default:

```
/**
 * {@inheritDoc}
 */
@Override // (or on a constructor)
```

The only case where {@inheritDoc} is useful is when you actually have additional Java documentation. Default JavaDoc interprets as replace the parent documentation. If you truly want the full text of the parent to be copy/pasted by JavaDoc in addition to your additional one, then use:

```
/**
 * {@inheritDoc}
 * For this specific implementation...
 */
@Override // (or on a constructor)
```

See also https://github.com/sevntu-checkstyle/sevntu.checkstyle/issues/467 & http://tornorbye.blogspot.ch/2005/02/inheriting-javadoc-comments.html

IllegalThrows

Instead of declaring "throws Exception", in almost all cases you should instead throw a custom existing or new ODL Exception. Instead of an unchecked exception (unchecked = extends RuntimeException; if you must for some technical reason, but should be rare, and avoided), it's recommended to use a custom module specific checked exception (checked = extends Exception); which can wrap a caught RuntimeException, if needed.

In order to avoid proliferation of many kinds of checked Exception subtypes for various particular nitty gritty things which could possibly go wrong, note that it in ODL is perfectly OK & recommended to have just ONE checked exception for a small given functional ODL module (subsystem), and throw that from all of its API methods. This makes sense because a typical caller wouldn't want do anything different - what you are "bubbling up" is just that one of the operations which one module asked another ODL module to do couldn't be performed. This avoids having multiple throws for each exception in API methods, and having problems with extendibility due to having to add more exceptions to the "throws" declaration of API methods.

The exception for "throws Exception" may be a main() method where it's customary to let anything propagate to the CLI, or `@Test testSomething() throws Exception` where it is acceptable (Checkstyle does NOT flag this particular use of "throws Exception" in @Test methods).

IllegalCatch

The IllegalCatch violation should almost never be suppressed in regular "functional" code. Normal code should only catch specific sub classes of the checked Exception, and never any generic and/or unchecked exceptions.

In the old pre-Java 7 days, some people used "catch (Exception e)" to "save typing" as a shorthand for having to catch a number of possibly thrown types of checked exceptions declared with "throws" by a method within the try block. Nowadays, since Java 7, using a multi-catch block is the right way to do this. In addition to being "nicer" to read because it's clearer, much more importantly than, a multi-catch does not also accidentally catch RuntimeException, as catch (Exception e) would. Catching RuntimeException such as NullPointerException & Co. is typically not what the developer who used "catch (Exception e)" as shorthand intended.

If a catch (Exception e) is used after a try { } block which does not call any methods declaring that they may throw checked exceptions with their throws clause (perhaps not anymore, after code was changed), then that catch may really have been intended to catch any possible RuntimeException instead? In that case, if there exceptionally really is a particular reason to want to "do something and recover from anything that could possibly go wrong, incl. NullPointerException, IndexOutOfBoundsException, IllegalArgumentException, IllegalStateException & Co.", then it is clearer to just catch (RuntimeException e) instead of catch (Exception e). Before doing this, double check that this truly is the intention of that code, by having a closer look at code called within the try, and see if that called code couldn't simply be made more robust. Be particularly careful with methods declaring checked exceptions in their "throws" clause: if any matching exception is thrown inside the "try" block, changing "catch (Exception e)" to "catch (RuntimeException e)" could change the method behavior since the exception will exit the method instead of being processed by the "catch" block.

Proliferation of catch (Exception or RuntimeException e) { LOG.error("It failed", e); } in regular "functional" code is a symptom of a missing abstraction in framework code; e.g. an Abstract interface implementation helper with correct default error handling, so that functional code does not have to repeat this over and over again. For example:

1. For DataBroker related transaction management, check out the (at the time of writing still in review) new utilities from c/63372 & Co.
2. For RPC related catch, see c/63413
3. Instead of `catch(Exception)` after a `try { close(anAutoCloseable) }` just use `AutoCloseables.closeOrWarn(anAutoCloseable)` introduced in https://git.opendaylight.org/gerrit/#/c/44145/

Sometimes developers also simply don't see that an existing framework API intends implementations to simply propagate their errors up to them. For example, for Exception handling in:

1. OsgiCommandSupport's `doExecute()`, the right thing to do is... nothing. The parent `doExecute()` method declaration throws Exception, and that is intentional by the Good People of Karaf. Therefore, `catch(Exception)` in a OsgiCommandSupport's `doExecute` is not required : in this case it's perfectly OK to just let any error "propagate" upwards automatically. If `doExecute()` calls other private methods of an OsgiCommandSupport implementation, then it is perfectly OK to make those methods declare `"throws SomeException"` too, and not "handle" them yourself.
2. Callable's `call()` passed to a `DataStoreJobCoordinator enqueueJob()`, the right thing to do is... nothing. Do not catch `(Exception)` but let it propagate. If it's useful to "augment" the exception message with more custom details which are available inside Callable's `call()`, then the right thing to do is to `catch (Exception e) { throw new YourProjectApiException("Failed to ... for {}", aDetail, e); }` and, exceptionally, use `@SuppressWarnings("checkstyle:IllegalCatch")`.
3. `org.opendaylight.infrautils.inject.AbstractLifecycle`'s start() and stop() methods, again the right thing to do is... nothing. Do not catch any Exception but let it propagate.

Here are some cases where `catch(Exception)` is almost always wrong, and a respective `@SuppressWarnings` almost never acceptable:

1. In Tests code you typically just `@Test testSomething() throws (Some)Exception` instead of catch, or uses `@Test(expected = ReadFailedException.class)`. One rare case we have seen where it's jus-

tified is a `@Test(expected = ReadFailedException.class)` with `catch (Exception e) throw e.getCause()`.

2. In one time "setup" (initialization) kind of code. For example, catch for a `DataBroker registerDataChangeListener` makes little sense - it's typically much better to let a failure to register a data change listener "bubble up" then continue, even if logged, and have users wonder why the listener isn't working much later.

Only in lower-level "Framework" kind of code, catch (Exception e) is sometimes justified / required, and thus `@SuppressWarnings("checkstyle:IllegalCatch")` acceptable.

Catching `Throwable` in particular is considered an absolute No No (see e.g. discussion in https://git.opendaylight.org/gerrit/#/c/60855/) in almost all cases. You may got confused any meant to catch Exception (see above) or RuntimeException?

Carefully consider whether you mean to catch and set some flag and/or log, and then rethrow, or intended to "swallow" the exception.

`System.out`

The `RegexpSingleLineJava` "Line contains console output" and "Line contains `printStackTrace`" should NEVER be suppressed.

In `src/main` code, `System.out.println` has no place, ever (it should probably be a `LOG.info`; and `System.err` probably a `LOG.error`).

In Java code producing Karaf console output, we should only use `System.out`, and add the corresponding `@SuppressWarnings`. `System.out` handles pipes and remote sessions correctly.

In `src/test` code, there should be no need to write things out - the point of a test is to assert something. During development of a test it is sometimes handy to print things to the console to see what is going on instead of using the debugger, but this should be removed in final code, for clarity, and non-verbose test execution. If you must, do you a Logger even in a test - just like in `src/main` code. This also makes it easier to later move code such as helper methods from test to production code.

Javadoc Paragraph: < p > tag should be preceded with an empty line

Checkstyle (rightfully) flags this kind of JavaDoc up as not ideal for readability:

```
/**
 * Utilities for...
 * <p>This...
```

and you can address this either like this:

```
/**
 * Utilities for...
 *
 * <p>This...
```

or like this:

```
/**
 * Utilities for...
 * <p/>
 * This...
```

Different ODL developers agree to disagree on which of the above is more readable.

Additional Resources

- `Checkstyle` http://checkstyle.sourceforge.net/
- `Maven`: https://github.com/checkstyle/checkstyle/blob/master/src/main/resources/google_checks.xml
- `Eclipse`: https://github.com/google/styleguide/blob/gh-pages/eclipse-java-google-style.xml
- `IntelliJ`: https://github.com/google/styleguide/blob/gh-pages/intellij-java-google-style.xml
- How to set Checkstyle up in IntelliJ old wiki page

Classes methods / fields ordering

Ordering based on modifiers. This is based on visibility and mutability:

public static final fields
static final fields
private static final fields
private final fields
private non-final fields
private volatile fields
private constructors
public constructors
static factory methods
public methods
static methods
private methods
The first group should be very strict, with the exception of FieldUpdaters, which should be private static final, but defined just above the volatile field they access. The reason for that is they are tied via a string literal name.

The second group is less clear-cut and depends on how instances are created – there are times when juggling the order makes it easier to understand what is going on (e.g. co-locating a private static method with static factory method which uses it).

The third group is pretty much free-for-all. The goal is to group things so that people do not have scroll around to understand the code flow. Public methods are obviously entry-points, hence are mostly glanced over by users.

Overall this has worked really well so far because;

- the first group gives a 10000-foot overview of what is going on in the class, its footprint and references to other classes
- the second group gives instantiation entry-points, useful for examining who creates the objects and how
- the third group is implementation details – for when you really need to dive into the details.

Note this list does not include non-private fields. The reason is that public fields should be forbidden, as should be any mutable non-private fields. The reason for that is they are a nightmare to navigate when attempting to reason about object life-cycle.

Same reasoning applies to inner class, keep them close to the methods which use them so that the class is easy to read. If the inner class needs to be understood before the methods that operate on it, place it before them, otherwise (especially if it's an implementation detail) after them. That's when an inner class is appropriate of course.

error-prone

The infrautils projects has adopted the error-prone code quality tool (by Google), with suitable customized configuration.

You can use it by using `org.opendaylight.infrautils:parent` instead of `org.opendaylight.odlparent:bundle-parent`.

Note that `@SuppressWarnings("InconsistentOverloads")` needs to be placed on the class, not method; see https://github.com/google/error-prone/pull/870 and https://github.com/google/error-prone/issues/869.

SpotBugs

SpotBugs is the successor project to FindBugs (which is dead).

SpotBugs is enforced by default across all artifacts since Magnesium. For artifacts that do not pass SpotBugs, either fix them or disable enforcement by defining the following property in the pom.xml:

```
<properties>
 <odlparent.spotbugs.enforce>false</odlparent.spotbugs.enforce>
</properties>
```

All notes re. FindBugs listed below do still apply to SpotBugs as well (it's compatible).

FindBugs

Note that starting with odlparent 3.0.0 when we say "FindBugs" we now actually mean "SpotBugs", see above.

`@FBSuppressWarnings`

If really needed, projects can to override individual FindBugs rules on a case-by-case basis by using a `@SuppressFBWarnings` annotation:

```
@SuppressFBWarnings("RCN_REDUNDANT_NULLCHECK_OF_NONNULL_VALUE")
public V get(K key) {
```

Unchecked/unconfirmed cast from `com.google.common.truth.Subject` to `com.google.common.truth.BooleanSubject` etc.

FindBugs seems to be too dumb to deal with perfectly valid Google Truth test code (which does not use any explicit cast…) so it's OK to:

```
@SuppressFBWarnings("BC_UNCONFIRMED_CAST")
```

an entire JUnit *Test class because of that.

null analysis errors, incl. FindBugs' NP_NONNULL_FIELD_NOT_INITIALIZED_IN_CONSTRUCTOR

see the general null analysis next chapter.

nonNullAndOptional

Some of the content from this chapter may be moved to http://www.lastnpe.org later...

Everything @NonNullByDefault

Do not use null anywhere, assume all method arguments and return values are NonNullByDefault.

null annotations from org.eclipse.jdt.annotation instead of javax.annotation

We prefer using the null annotations from the package org.eclipse.jdt.annotation , instead of the ones from javax.annotation (or those from org.jetbrains:annotations, or ... Android/Lombok's/some of the other ones out there).

This is because the org.eclipse one are modern generics enabled @Target TYPE_USE annotations, whereas the other ones are not.

Obviously we do NOT "depend on Eclipse" in any way, or "need Eclipse at run-time" just because of 4 annotations from an org.eclipse package, which are available in a very small JAR at build-time; e.g. org.eclipse.jdt.annotation.NonNull is absolutely no different from javax.annotation.Nullable, in that regard.

BTW: The javax.annotation NonNull & Co. are not more or less "official" than others; Prof. FindBugs Bill Pugh pushed those to Maven central, but his "dormant" JSR 305 was never officially finalized and approved; he's since moved on, and no longer maintains FindBugs.

The Eclipse annotations are also supported by SpotBugs/FindBugs (says this issue) as well as NullAway.

null analysis by FindBugs VS. Eclipse JDT

FindBugs' null analysis is inferior to Eclipse JDT's, because:

- richer null analysis
- generics enabled (see above)
- works with existing external libraries, through external annotations, see http://www.lastnpe.org
- much better in-IDE support, at least for Eclipse users

*WIP: We are aiming at, eventualy, running headless Eclipse JDT-based null analysis during the build, not just for users of the Eclipse IDE UI; watch*issue ODLPARENT-116 *, and*http://www.lastnpe.org.

BTW: FindBugs is dead anyway, long live SpotBugs! _TODO Gerrit & more documentation to clarify this..._

PS: An alternative interesting null checker tool is the Checker Framework.

Runtime null checks

In addition to static null analysis during development, you can check null safety at run-time. Please use either JDK's Object's requireNonNull () or Guava's Preconditions `checkNotNull()` utility, instead of if (something == null). Please also use the variant of `requireNonNull()` or `checkNotNull()` with the String message to indicate what argument is checked. For example:

```
public doSomething(Something something) {
    this.something = Objects.requireNonNull(something, "something");
}
```

We recommend use of JDK's Object's `requireNonNull()` instead of Guava's Preconditions `checkNotNull()` just because the String message of the former can prevent the problem you can have with the latter if you confuse the order of the arguments.

We accept and think its OK that `checkNotNull()` throws a NullPointerException and not an IllegalArgumentException (even though code otherwise should never manually throw new NullPointerException), because in this particular case a NullPointerException would have happened anyway later, so it's just an earlier NullPointerException, with added information of what is null.

We NEVER catch (NullPointerException e) anywhere, because they are programming errors which should "bubble up", to be fixed, not suppressed.

`nullable` errors for fields related to dependency injection

null analysis frameworks, such as Eclipse's or FindBugs or other, will not like this kind of code in a `@NonNullByDefault` package:

```
class Example {
     @Inject
     Service s;
}
```

the recommended solution is to always use constructor instead of field injection instead, like this:

```
class Example {
    final Service s;
    @Inject
    Example(Service s) {
        this.s = s;
    }
}
```

When this exceptionally is not possible, like in a JUnit component test, then it's acceptable to suppress warnings:

```
@SuppressFBWarnings("NP_NONNULL_FIELD_NOT_INITIALIZED_IN_CONSTRUCTOR")
class SomeTest {
    public @Rule GuiceRule guice = new GuiceRule(TestModule.class);
    private @Inject Service service;
}
```

Optional

You do not have to use Optional, because real null analysis can give us the same benefit.

If cleaning up code for null safety, then do not mix introducing Optional with other null related clean up changes; it's clearer for reviews if you FIRST fix missing null checks and add null related annotations, and then THEN (optionally) switch some return types to Optional.

You can use Optional for return types, but not method arguments.

Never use `Optional<Collection<...>>` (obviously incl. `Optional<List<...>>` or `Optional<Set<...>>` AND `Optional<Map<...>>` etc.), just return a respective empty Collection instead.

Note that instead of if `(anOptional.isPresent()) { return anOptional.get(); } else { return null; }` you can and for readability should just use return `anOptional.orNull()`. However ideally any such code should not return null but an Optional of something itself.

Note that instead of `if (aNullable == null) ? Optional.absent() : Optional.of(aNullable)a` ;you can and for readability should just use `Optional.fromNullable(aNullable)`.

To transform an Optional into something else if it present, use the transform() method instead of an if () check;. for example:

```
List vrfEntries = MDSALUtil.read(broker, LogicalDatastoreType.CONFIGURATION,␣
↪vpnVrfTables).transform(VrfTables::getVrfEntry).or(new ArrayList<>());
```

Take care with `Optional.transform()` though: if the transformation can return null, `Optional.transform()` will fail with a NullPointerException (this is the case of most YANG-generated methods). You can use Java 8 `Optional.map()` instead; when it encounters null, it returns `Optional.empty()`. The above example would become

```
List<VrfEntry> vrfEntries = MDSALUtil.read(broker, LogicalDatastoreType.CONFIGURATION,␣
↪vpnVrfTables).toJavaUtil().map(VrfTables::getVrfEntry).orElse(new ArrayList<>());
```

Prefer the newer Java 8 `java.util.Optional` (JUO) over the older Google Guava `com.google.common.base.Optional` (GGO), because it offers a better functional style API for use with Java 8 lambdas, which leads to much more naturally readable code. Until we fully migrate all ODL APIs (which is planned), in glue code calling existing APIs returning GGO (such as the DataBroker API) but itself then wanting to return a function of that as JUO, please just use the `toJavaUtil()` method available in Guava Optional.

Further Reading & Watching

- https://github.com/google/guava/wiki/UsingAndAvoidingNullExplained
- https://stackoverflow.com/questions/26327957/should-java-8-getters-return-optional-type

Streaming and lambdas

Lambdas are very useful when encapsulating varying behavior. For example, you can use them instead of boolean behavior selection parameters:

```
public void someMethodA(SomeObject A) {
    commonMethod(A, false);
}

public void someMethodB(SomeObject B) {
```

(continues on next page)

(continued from previous page)

```
    commonMethod(B, true);
}

private void commonMethod(SomeObject C, boolean behaviour) {
    // common code

    if (behaviour) {
        doA();
    } else {
        doB();
    }

    // common code
}
```

can be replaced with

```
public void someMethodA(SomeObject A) {
    commonMethod(A, this::doA);
}

public void someMethodB(SomeObjectB) {
    commonMethod(B, this::doB);
}

private void commonMethod(SomeObject C, Function behaviour) {
    // common code

    behaviour.apply();

    // common code
}
```

They are also useful for replacing small anonymous inner classes which follow the functional pattern (implementing an interface with a single non-default method). Your IDE will typically suggest such transformations.

Lambdas should be avoided when the resulting code is more complex and/or longer than the non-functional form. This can happen particularly with streaming.

Streaming also has its place, especially when combined with Optional (see above) or when processing collections. It should however be avoided when many objects are created in the resulting lambda expressions, especially if DTOs end up being necessary to convey information from one lambda to the next where a single variable could do the trick in a more traditional form. (TODO: provide examples.)

Coding Guidelines for Python

Note: This document is a work in progress.

PEP8 is the Python standard that should be followed when coding any Python code with the following exceptions.

- 120 characters line length

To automate pep8 scanning we recommend using a `tox.ini` configuration file as follows:

```
[tox]
envlist = pep8
#skipsdist = true  # Command only available in tox 1.6.0

[testenv:pep8]
deps = flake8
commands = flake8

[flake8]
max-line-length = 120
```

Unfortunately the version of tox installed in the Jenkins build infra does not support the `skipdist` parameter which is necessary if your project does not have a `setup.py` file. A workaround is to create a minimal `setup.py` file as follows:

```
# Workaround for tox missing parameter 'skipsdist=true' which was not
# introduced until tox 1.6.0

import setuptools

setuptools.setup(name='project-name')
```

Coding Guidelines for XML

Note: This document is a work in progress.

- use self-closing
- include proper namespace/model/version declarations
- TBD

Coding Guidelines for YANG

Note: This document is a work in progress.

- Do not use underscores ('_') in identifiers. JDK 9 is on track to making underscores forbidden in identifiers, which means we will need to map them and it is not going to be pleasant :-(
- Each declaration needs to have either a description or a reference to a definition document (like an IETF draft)
- Use `typedefs` to declare concepts. An UUID is typeless, so each instance should have its scope, so we know its applicability domain. 'type string' should only be used to things like free-form comments and similar. Please attach a 'units' statement whenever possible.
- TBD

Jira Ticketing Guideline

Mandatory Fields

Here is the list of mandatory fields for each Jira ticket:

Project:

description: the name of related project

values: aaa, bgpcep, controller, etc.

Priority:

description: the importance of ticket

values: Highest (reserved for release blocker bugs), High, Medium, Low, Lowest

Type:

description: issue type

values: Bug/New Feature/Task/Improvement/Deprecate

affectedVersion:

description: the version that the issue was seen (it's only applicable to bugs)

values: aluminum/silicon/etc.; "None" to be used for anything other than "Bug"

fixVersion:

description: the version targeted for the bug fix, new feature, behavior/feature change (improvement), or deprecated feature (the first time that the feature is being disappeared), the release schedule can be found here

values: aluminum/silicon/etc.

About Copyright and License

Note: This document is a work in progress.

Contributors must comply with the terms of the OpenDaylight Technical charter.

Following are the requirements with regards to Copyright and License statements in all source code in the OpenDaylight projects.

New Files

Every file should contain a copyright statement, which at a minimum states that the source code is available under the Eclipse Public License (EPL) or a compatible license. By placing the copyright in all source files, downstream consumers of OpenDaylight can refer to their redistribution clauses.

The OpenDaylight Project does not request or require copyright assignment to the OpenDaylight non-profit organization, so adding the original contributor's name or company is allowed.

A missing copyright statement is treated as a bug so the EPL license can at least be added for downstream consumers.

Possible structure of the EPL copyright header:

```
/**
 * Copyright (c) <Date> <Company or Individual> and others.  All rights reserved.
 *
 * This program and the accompanying materials are made available under the
 * terms of the Eclipse Public License v1.0 which accompanies this distribution,
 * and is available at http://www.eclipse.org/legal/epl-v10.html
 **/
```

Note: If you are contributing to OpenDaylight on behalf of a company or organization, please check with your company's legal department if you need to use your company name's or if you can use your personal name as the copyright holder. Be sure to place the current date and company name in the copyright. If copying a copyright please make sure that you place the correct company's name into the copyright (i.e. if you copy an existing copyright statement, make sure you change the name to give credit to the correct people or organizations!).

Edited Files

If you, the editor, feel like you have made a *significant* contribution for which you want to issue a copyright, then modify the copyright, adding your company's name or individual name into the original header.

```
/**
 * Copyright (c) <Date> Existing Company One, and others.  All rights reserved.
 * Copyright (c) <Date> New Company Two
 *
 * This program and the accompanying materials are made available under the
 * terms of the Eclipse Public License v1.0 which accompanies this distribution,
 * and is available at http://www.eclipse.org/legal/epl-v10.html
 **/
```

In the example above, "New Company Two" was added into the copy right. "Existing Company One" already existed and remained in the copyright header. Whether or not you edit the existing copyright is up to you.

Note: Never remove or replace the copyright on a file. The original author still retains the copyright on the original work. You retain the copyright on the new edits only. *Significant* is purposely left vague to be left to your best judgement. At the end of the day if the issue of who owns what changes ever arises it will be left to the lawyers and judges to determine the copyright ownership. Remember, all changes are logged in the git repository.

- LFN Gerrit Guide