


- 
- 
- Page
  - Page
  - Discussion
  - View source
  - History
- Tools
  - Page information
  - Permanent link
  - Printable version
  - 
  - Special pages
  - Related changes
  - 
  - What links here

- 
- 
- 
- Log in
- OpenDaylight.org
- Get Software
- Documentation
- User Stories
- Community
- Blog
- Q&A Form
- Wiki

## OpenDaylight Controller:MD-SAL:Startup Project Archetype

### Contents

- 1 Introduction
- 2 Setup
- 3 Part 1 – Build with a simple 'Example' module
  - 3.1 Ok it's all downloaded....now what?
  - 3.2 Time to build the 'example' project
  - 3.3 Starting your 'example' project for the first time
  - 3.4 Check for basic functioning

- 4 Part 2 – Hello World – Defining a Simple RPC
  - 4.1 Run the archetype and create the 'hello' project
  - 4.2 Have a look at 'hello' project
  - 4.3 Build 'hello' project using:
  - 4.4 Check for basic functioning
  - 4.5 Understanding where the log line came from (the entry point)
  - 4.6 Adding a very simple HelloWorld RPC api
  - 4.7 Implementing the HelloWorld RPC API
  - 4.8 Running your 'hello' project for the first time
- 5 Testing the 'hello-world' RPC via REST
  - 5.1 Using the API Explorer via http.
  - 5.2 Using a browser REST client
- 6 Troubleshooting
- 7 Next Steps

## Introduction

This tutorial is aimed at a developer who would like a quick start to OpenDaylight Development. It will create a local repository for the code and guide you through a simple build process. Once the tutorial is complete you will be able to start OpenDaylight and test a simple RPC. The test RPC will be one you have created, based on the principle of 'hello world'.

Ideally you will have already have a working knowledge of Maven 3.2.1 (at time of writing), Java 1.8.0\_60 (at time of writing) and how to set up an environment with the correct paths/variables on your development machine. The GettingStarted: Eclipse automated set up is a convenient way to get started.

During this tutorial you will use the Maven Archetype framework, which provides a template approach to downloading projects. For more details please read <http://maven.apache.org/guides/introduction/introduction-to-archetypes.html> .

## Setup

Once you have setup your development machine with Maven and Java, you will need to update your Maven settings.xml. If you are using GettingStarted: Eclipse, then this is automatically set up for you. Otherwise, do this by editing your settings.xml file. (You should view this file to get a better understanding as to what it is for, which is to define the repositories for this project.)

**IMPORTANT:** If you already have previously downloaded OpenDaylight for another project it is recommended that you remove an existing repository before you start this project. If you are simply rebuilding the projects below, having previously run this tutorial, then you don't need to delete your local repository. (If you are using Linux on as your development machine, you will find your local repository in ~/.m2/repository. Other platforms may vary.)

If you are using Windows, please make sure Microsoft Visual C++ environment is correctly installed, otherwise opendaylight will quit with error.

## Part 1 – Build with a simple 'Example' module

Let's begin by creating a simple 'Example' project using Maven and an archetype called 'opendaylight-startup-archetype'.

If this is the first time you are downloading this project then it will take a while to pull all the code from the remote repository.

NOTE: If it is failing below, remove ~/.m2 directory. Also make sure you have the OpenDaylight setting.xml file as described above.

If you are using the GettingStarted: Eclipse fully automated set up, then you can now use menu File > New > Project > Maven > Maven Project, Next; Advanced: Name template: [groupId]. [artifactId], and then check [X] Include snapshot archetypes, and choose Artifact Id opendaylight-startup-archetype. (Note that opendaylight-eclipse-setup already has a catalog with the required archetypeRepository from ODL.)

Otherwise, create your project with the archetype by typing:

```
mvn archetype:generate -DarchetypeGroupId=org.opendaylight.controller -DarchetypeArtifactId=opendaylight-startup-archetype -DarchetypeRepository=http://nexus.opendaylight.org/content/repositories/<Snapshot-Type>/ \
-DarchetypeCatalog=remote -DarchetypeVersion=<Archetype-Version>
```

You need to enter the proper <Archetype-Version> and <Snapshot-Type> that depends on the ODL release you want to work in. for example:

- For the current Master (Carbon) use Snapshot-Type=**opendaylight.snapshot** Archetype-Version=**1.4.0-SNAPSHOT**
- For the Carbon snapshot use Snapshot-Type=**opendaylight.release** Archetype-Version=**1.3.0-Carbon**
- For Boron "SR0" use Snapshot-Type=**opendaylight.release** Archetype-Version=**1.2.0-Boron**
- For Boron SR1 use Snapshot-Type=**opendaylight.release** Archetype-Version=**1.2.1-Boron-SR1**
- For Boron SR2 use Snapshot-Type=**opendaylight.release** Archetype-Version=**1.2.2-Boron-SR2**
- For the Boron snapshot use Snapshot-Type=**opendaylight.snapshot** Archetype-Version=**1.2.2-SNAPSHOT**

Note each version of the archetype generates version numbers in pom.xml dependencies for its intended ODL revision.

Respond to the prompts (Please note that groupId and artifactId need to be all lower case):

```
Define value for property 'groupId': : org.opendaylight.example
Define value for property 'artifactId': : example
Define value for property 'package': : org.opendaylight.example:
Define value for property 'classPrefix': : ${artifactId.substring(0,1).toUpperCase()}${artifactId.substring(1)}
Define value for property 'copyright': : Yoyodyne, Inc.
```

In particular, accept the default value of classPrefix (\$\${artifactId.substring(0,1).toUpperCase()}\${artifactId.substring(1)}) which creates a Java Class Prefix by capitalizing the first character of the artifactId ( so in this example the classPrefix will be Example). If you want to change any of the defaults, say 'N' at the last question; which will give you the opportunity to change them.

## Ok it's all downloaded....now what?

The archetype will have create a top level directory:

```
${artifactId}/
```

in our example:

```
example/
```

Enter the directory:

```
cd example/
```

and look around:

```
api/  
artifacts/  
features/  
impl/  
karaf/  
pom.xml
```

## Time to build the 'example' project

Let's build the project for the first time. Depending on your development machine's specification this might take a little while. Ensure that you are in the project's root directory, example/, and then issue the build command, shown below.

```
mvn clean install
```

## Starting your 'example' project for the first time

Once the project has built you will have created an Opendaylight distribution. Change to the directory and have a look.

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

Once you are ready start the 'example' project using the following command.

```
./karaf
```

Wait for the karaf cli:

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

**IMPORTANT:** Wait for OpenDaylight to fully load all the components which can take a minute or two after the prompt appears. (check the CPU on your dev machine, specifically the Java process, to see when it calms down.

## Check for basic functioning

During the build process a module called 'Example' was built, which you can now verify on the console by checking out the log:

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

Look for the log entry which includes the entry 'ExampleProvider Session Initiated'.

To shutdown OpenDaylight via the console issue the command below.

```
shutdown -f
```

## Part 2 – Hello World – Defining a Simple RPC

In part 2 creates a new project, again using the Maven archetype 'opendaylight-startup-archetype', but this time creating a 'hello' project.

It would be beneficial to have a working knowledge of Java development, since Part 2 of this wiki explains at how to augment the 'hello' project to build the example API.

If you have already done the 'example' project the time to pull the archetype is greatly reduced, since you now have a local repository in ~/.m2/repository (in Linux other systems may vary for location). If you have deleted the local repository, Maven will re-fetch it.

## Run the archetype and create the 'hello' project

```
mvn archetype:generate -DarchetypeGroupId=org.opendaylight.controller -DarchetypeArtifactId=opendaylight-startup-archetype -DarchetypeRepository=http://nexus.opendaylight.org/content/repositories/opendaylight.snapshot/ \ -DarchetypeCatalog=http://nexus.opendaylight.org/content/repositories/opendaylight.snapshot/archetypes
```

Respond to the prompts:

```
Define value for property 'groupId': : org.opendaylight.hello
Define value for property 'artifactId': : hello
Define value for property 'version': 1.0-SNAPSHOT
Define value for property 'package': org.opendaylight.hello: :
Define value for property 'classPrefix': hello
Define value for property 'copyright': : Yoyodyne, Inc.
```

## Have a look at 'hello' project

```
cd hello/
ls -l
api
artifacts
```

```
features
impl
karaf
pom.xml
```

## Build 'hello' project using:

```
mvn clean install
```

## Check for basic functioning

Run karaf:

```
cd karaf/target/assembly/bin
and then execute
./karaf
```

Wait for the karaf cli:

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

IMPORTANT: Remember to wait for OpenDaylight to fully load. (check Java process CPU has calmed down).

Verify the 'hello' module has loaded by checking out the log:

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

Shutdown karaf

```
shutdown -f
```

Return to the top of the directory structure:

```
cd ../../../../
```

## Understanding where the log line came from (the entry point)

The entry point is in the impl project:

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

In the HelloProvider.init method:

```
public void init() {  
    LOG.info("HelloProvider Session Initiated");  
}
```

This is the method you would add any new things you are doing in your implementation. It is analogous to an Activator.

## Adding a very simple HelloWorld RPC api

Edit

```
api/src/main/yang/hello.yang
```

Edit this file to look as below. You will see that we are adding the code in a YANG module to define the 'hello-world' RPC:

```
module hello {  
    yang-version 1;  
    namespace "urn:opendaylight:params:xml:ns:yang:hello";  
    prefix "hello";  
  
    revision "2015-01-05" {  
        description "Initial revision of hello model";  
    }  
    rpc hello-world {  
        input {  
            leaf name {  
                type string;  
            }  
        }  
        output {  
            leaf greeting {  
                type string;  
            }  
        }  
    }  
}
```

If you are using the GettingStarted: Eclipse set up, then you can now use right-click on the project, and use the context menu Run As > Maven generate-sources. Otherwise, return to the hello/api directory and build your api:

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

## Implementing the HelloWorld RPC API

We now need to define the HelloService, which will be called via the 'hello-world' API.

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

Create a new file called HelloWorldImpl.java and add in the code below.

```
package org.opendaylight.hello.impl;  
  
import java.util.concurrent.Future;
```

```

import org.opendaylight.yang.gen.v1.urn.opendaylight.params.xml.ns.yang.hello.rev150105.HelloService;
import org.opendaylight.yang.gen.v1.urn.opendaylight.params.xml.ns.yang.hello.rev150105.HelloWorldInput;
import org.opendaylight.yang.gen.v1.urn.opendaylight.params.xml.ns.yang.hello.rev150105.HelloWorldOutput;
import org.opendaylight.yang.gen.v1.urn.opendaylight.params.xml.ns.yang.hello.rev150105.HelloWorldOutputBuilder;
import org.opendaylight.yangtools.yang.common.RpcResult;
import org.opendaylight.yangtools.yang.common.RpcResultBuilder;

public class HelloWorldImpl implements HelloService {

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

```

In order to register our RPC with MD-SAL, we first need to create the reference to the RPC Registry. This is done in a couple of steps. First, we need to add the RPC Registry reference to the *impl-blueprint.xml* file in *src/main/resources/org.opendaylight/blueprint* as follows:

- We add a line that defines the reference to the implementation of the RPC Registry interface (*org.opendaylight.controller.sal.binding.api.RpcProviderRegistry*). We call the reference *rpcRegistry*.
- We add the reference to the implementation of the RPC Registry interface to the parameters of the "HelloProvider" constructor.

```

<?xml version="1.0" encoding="UTF-8"?>
<!-- vi: set et smarttab sw=4 tabstop=4: -->
<!--
Copyright © 2016 Cisco Systems 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">

  <reference id="dataBroker"
    interface="org.opendaylight.controller.md.sal.binding.api.DataBroker"
    odl:type="default" />

  <reference id="rpcRegistry"
    interface="org.opendaylight.controller.sal.binding.api.RpcProviderRegistry"/>'''

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

</blueprint>

```

Next, we add the reference to the RPC registry to the *HelloProvider* class and allow its injection into the class through the *HelloProvider*'s constructor:

```

/*
 * Copyright © 2016 Cisco Systems 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 org.opendaylight.controller.md.sal.binding.api.DataBroker;
import org.opendaylight.controller.sal.binding.api.RpcProviderRegistry;
import org.opendaylight.controller.sal.binding.api.BindingAwareBroker.RpcRegistration;
import org.opendaylight.yang.gen.v1.urn.opendaylight.params.xml.ns.yang.hello.rev150105.HelloService;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class HelloProvider {

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

    private final DataBroker dataBroker;
    private final RpcProviderRegistry rpcProviderRegistry;

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

    /**
     * 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");
    }
}

```

Finally, in the HelloProvider's init() function (called when the blueprint container is created) we register the HelloService RPC with MD-SAL as follows:

```

....
private RpcRegistration<HelloService> serviceRegistration;
...

/**
 * Method called when the blueprint container is created.
 */
public void init() {
    serviceRegistration = rpcProviderRegistry.addRpcImplementation(HelloService.class, new HelloWorldImpl());
    LOG.info("HelloProvider Session Initiated");
}

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

```

The next step is optional, however you can build just the java classes which will register the new RPC. This is useful to test the edits you have made to HelloProvider.java and HelloWorldImpl.java

```

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

```

Return to the top level directory

```
cd ../
```

Now build the entire 'hello' again, which will pickup the changes you have made and build them into your project:

```
mvn clean install
```

## Running your 'hello' project for the first time

Run karaf:

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

As before, wait for the project to completely load. Then view the log to see the loaded 'Hello' Module:

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

## Testing the 'hello-world' RPC via REST

There are a lot of ways to test your RPC. The ones listed below are some examples.

### Using the API Explorer via http.

Go to the RestConf API Documentation UI (Swagger/Open API Initiative (OAI)-based) (<http://localhost:8181/apidoc/explorer/index.html>) with your web browser. You will be prompted for a username and password, by default they are admin/admin.

NOTE – in the above URL you might need to change 'localhost' to the IP/Host name to reflect your development machine's network address.

Click on

```
hello(2015-01-05)
```

and then click on

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

and provide value:

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

and click the button

```
Try it out
```

In the response body you should see.

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

## Using a browser REST client

For example the Firefox plugin 'RESTClient' [1] (<https://github.com/chao/RESTClient>) or the Chrome &

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

```
Header:
Content-Type: application/json
```

```
Body:
{"input": {
  "name": "Giles"
}}
```

## Troubleshooting

If while attempting to POST /operations/hello:hello-world you get a response code 501 check the file: HelloProvider.java and make sure the helloService member is being set. By not invoking "session.addRpcImplementation()" the REST api will be unable to map the /restconf/operations/hello:hello-world URL to the HelloWorldImpl Java class.

## Next Steps

Check out the other tutorials!

Retrieved from "[https://wiki.opendaylight.org/index.php?title=OpenDaylight\\_Controller:MD-SAL:Startup\\_Project\\_Archetype&oldid=58761](https://wiki.opendaylight.org/index.php?title=OpenDaylight_Controller:MD-SAL:Startup_Project_Archetype&oldid=58761)"