OSGi workshop

Contents

Introduction

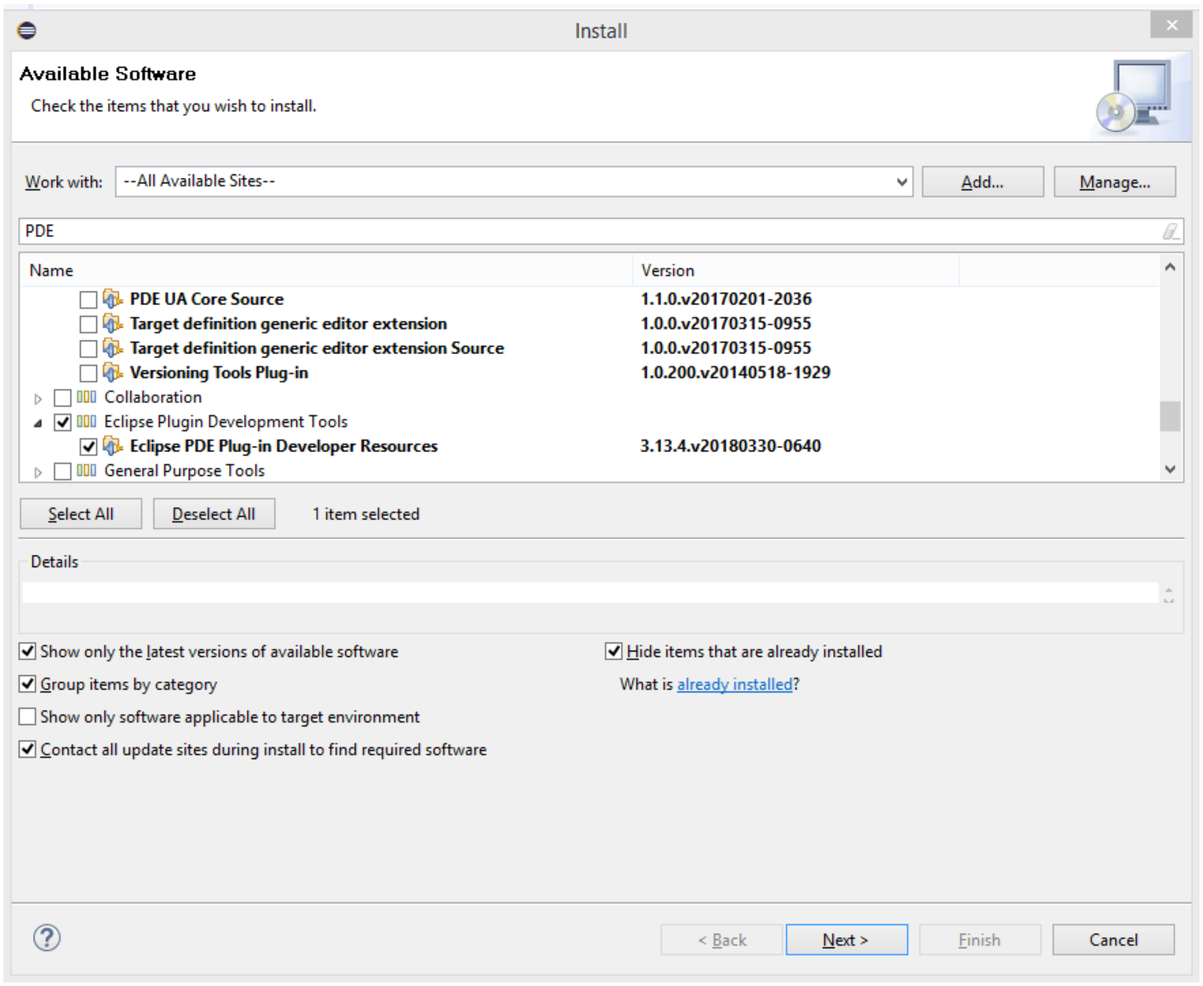
In this workshop you learn about the fundamentals of OSGi, how to implement, test and deploy OSGi modules. Estimated duration for the completion of the exercises is about 6-7 hours.

Prerequisites

Before you start with the exercises in this workshop you need to have the following installed:

- Eclipse for plug-in developers (with PDE support): https://www.eclipse.org/
- Apache Felix runtime: http://cxf.apache.org/download.html
- Apache Karaf: http://apache.cbox.biz/karaf/4.2.1/apache-karaf-4.2.1.zip
- Apache Maven: https://maven.apache.org/
- BND Tools: https://bndtools.org/installation.html

If you already have Eclipse installed but without PDE support you can additionally install it though **Help -> Install new software**, **specify All available sites** under **Work with** and in the search box type **PDE**. Then under **Eclipse Plugin Development Tools** specify **Eclipse PDE Plug-in Developer Resources**:


Install

Available Software

Check the items that you wish to install.

Work with: --All Available Sites--

PDE

Name	Version
☐ PDE UA Core Source	1.1.0.v20170201-2036
☐ Target definition generic editor extension	1.0.0.v20170315-0955
☐ Target definition generic editor extension Source	1.0.0.v20170315-0955
☐ Versioning Tools Plug-in	1.0.200.v20140518-1929
☐ Collaboration	
☑ Eclipse Plugin Development Tools	
☑ Eclipse PDE Plug-in Developer Resources	3.13.4.v20180330-0640
☐ General Purpose Tools	

1 item selected

☑ Show only the latest versions of available software

☑ Group items by category

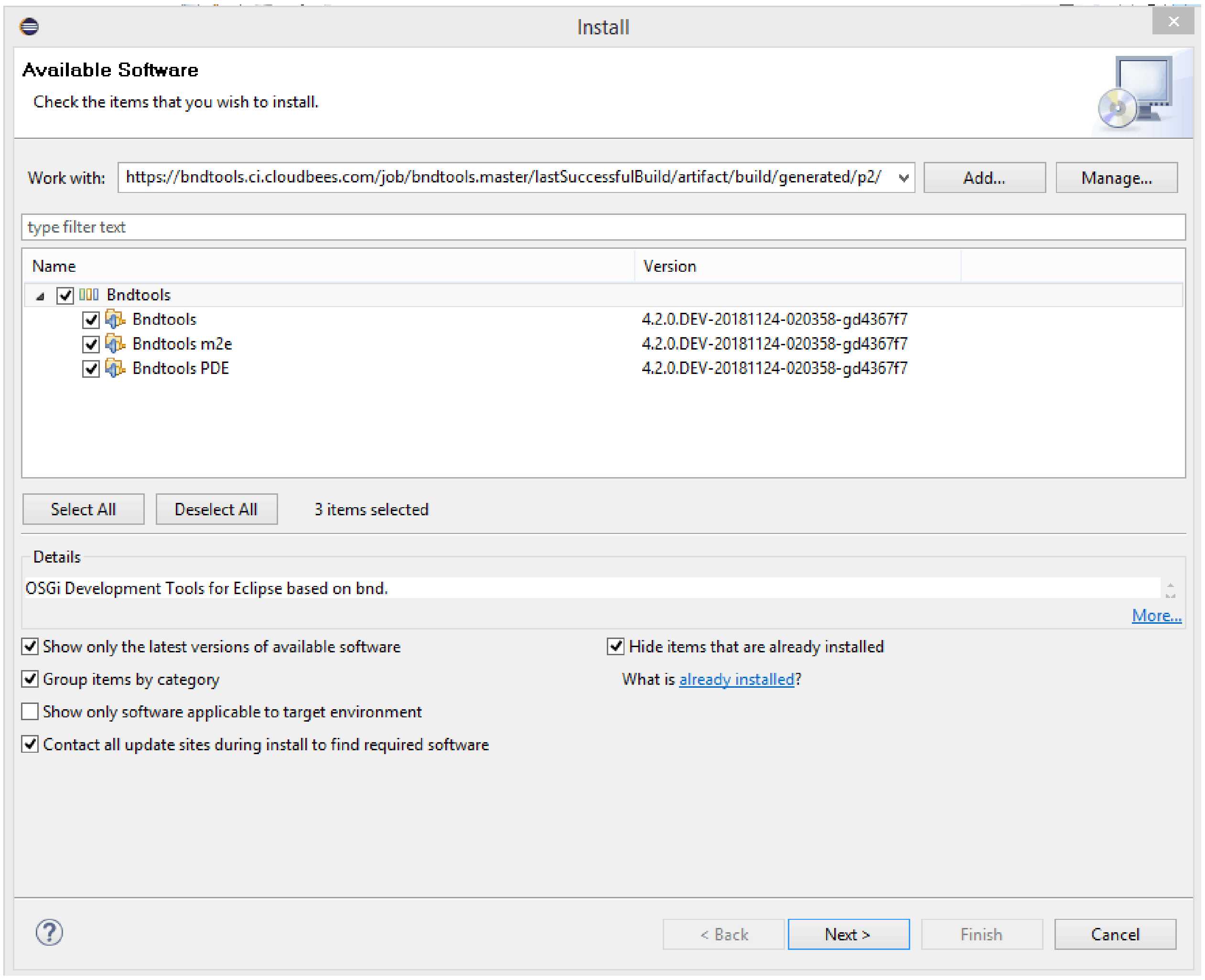
☐ Show only software applicable to target environment

☑ Contact all update sites during install to find required software

☑ Hide items that are already installed

What is already installed?

To install BND Tools in Eclipse you can use the https://bndtools.ci.cloudbees.com/job/bndtools.master/lastSuccessfulBuild/artifact/build/generated/p2/ update site and specify all BND components to install:


Install

Available Software

Check the items that you wish to install.

Work with: https://bndtools.ci.cloudbees.com/job/bndtools.master/lastSuccessfulBuild/artifact/build/generated/p2/

type filter text

Name	Version
☑ Bndtools	
☑ Bndtools	4.2.0.DEV-20181124-020358-gd4367f7
☑ Bndtools m2e	4.2.0.DEV-20181124-020358-gd4367f7
☑ Bndtools PDE	4.2.0.DEV-20181124-020358-gd4367f7

3 items selected

Details

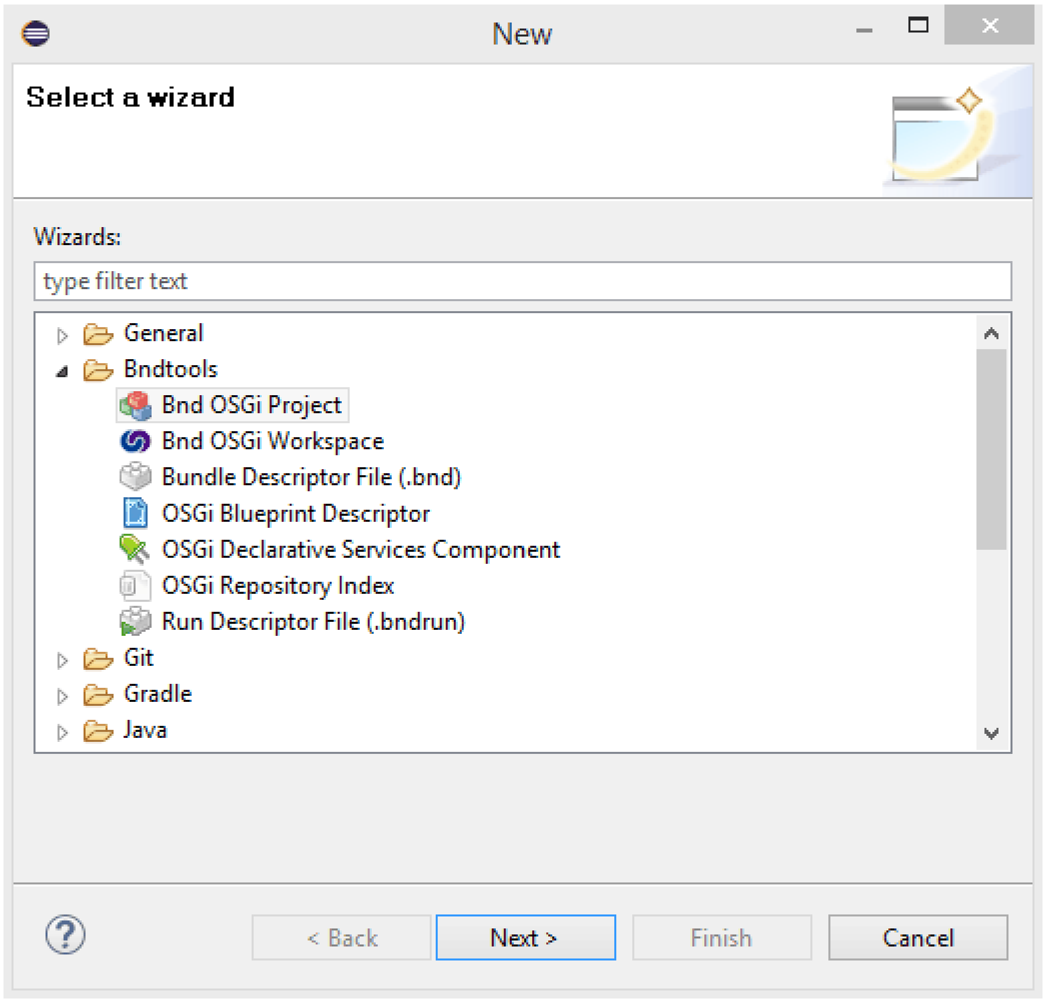
OSGi Development Tools for Eclipse based on bnd.

- ☑ Show only the latest versions of available software
- ☑ Group items by category
- ☐ Show only software applicable to target environment
- ☑ Contact all update sites during install to find required software
- ☑ Hide items that are already installed

What is already installed?

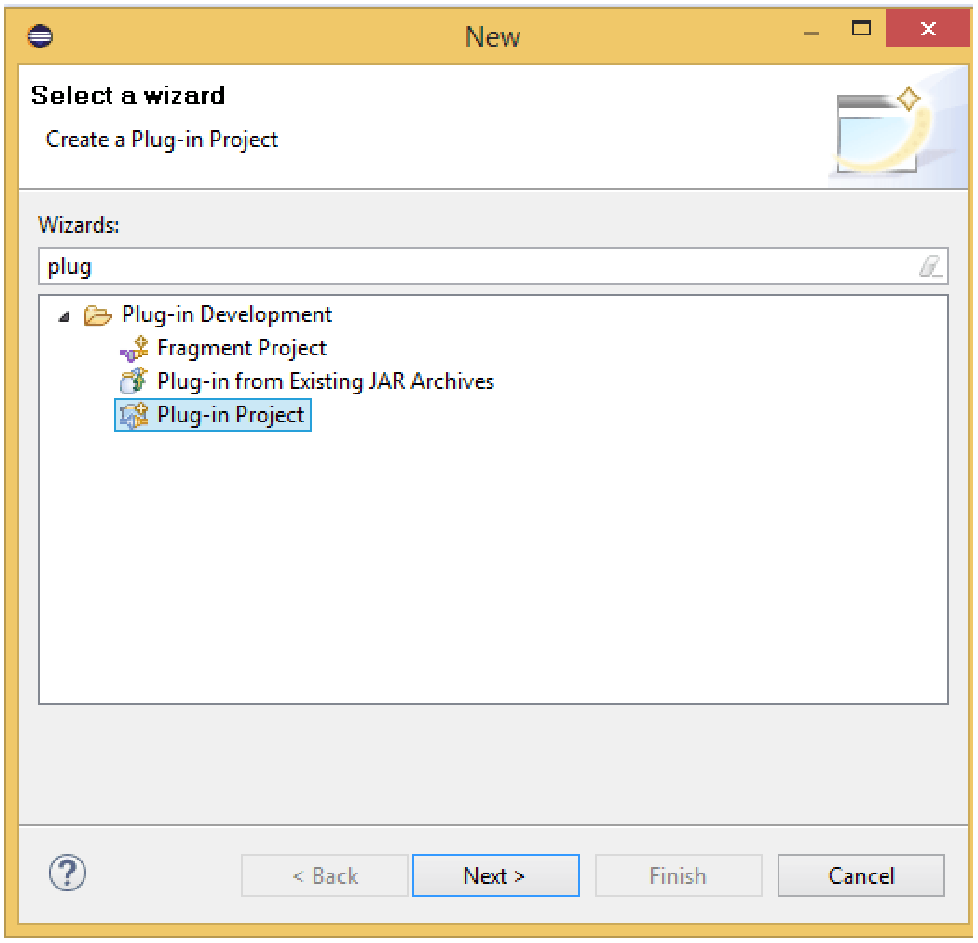
OSGi runtime basics

In the first exercise we will develop a simple OSGi bundle in Eclipse and deploy in either in an Equinox or a Felix runtime. Our first bundle will be used to provide a common interface for document generation bundles that convert text to a particular format (PDF, Word, txt etc.). In this workshop we will be creating plug-in projects which are OSGi bundles and can be also used to create Eclipse plug-ins. We will call our project **docgen** and it will be used to group the set of OSGi bundles (modules) we are going to create. You can also create OSGi bundles in Eclipse using BND tools from the new project wizard by specifying **Bnd OSGi Project** from the dialog:

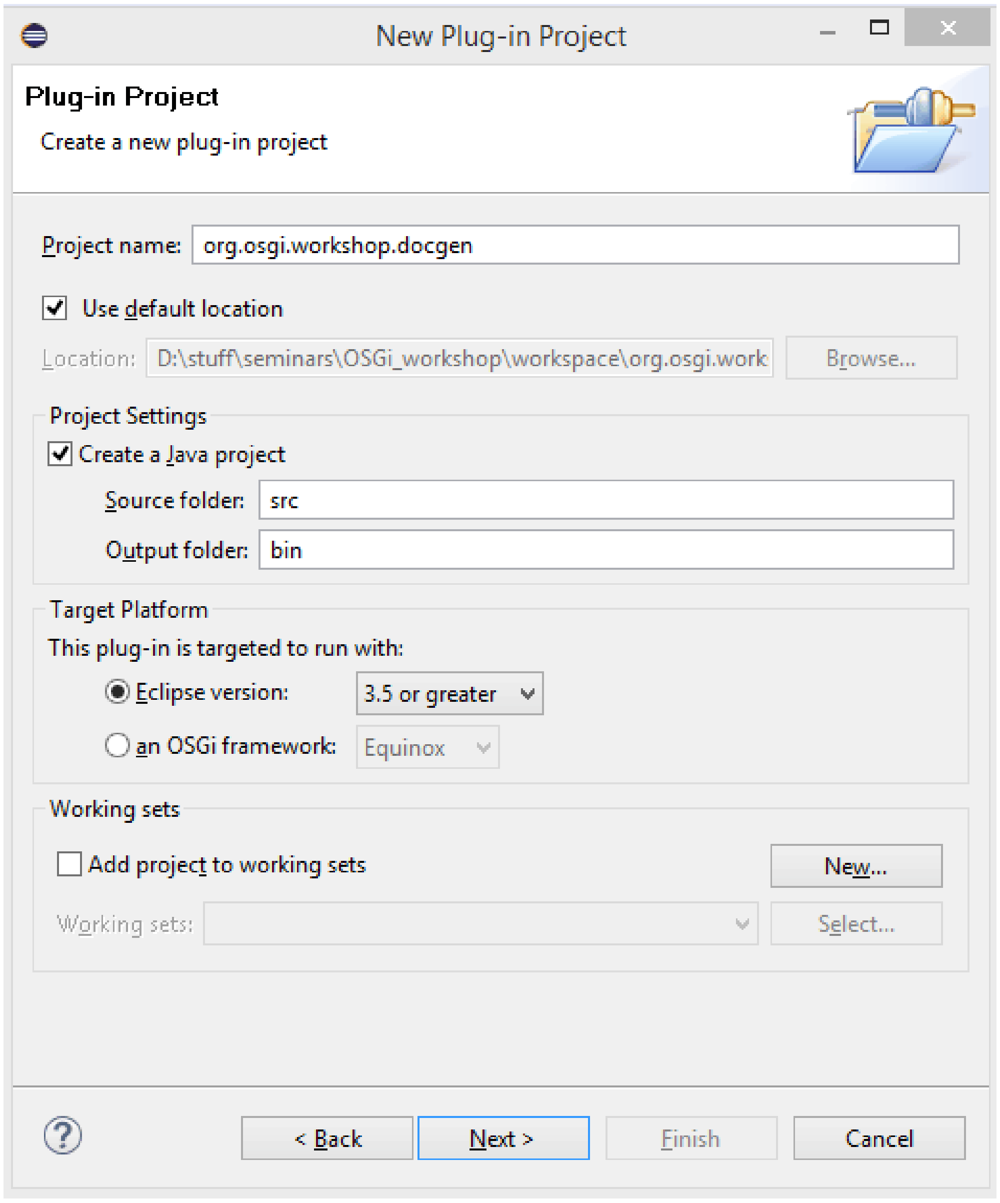


Later on in the workshop we will see how to use BND tools to simplify OSGi development but for now we will create our bundles using Plug-in projects.

Within Eclipse we will create our first bundle by specifying **File** -> **New** -> **Other** -> **Plug-in Project**:



For the project name specify **org.osgi.workshop.docgen** and leave the rest as specified:


New Plug-in Project

Plug-in Project

Create a new plug-in project

Project name: org.osgi.workshop.docgen

[✓] Use default location

Location: D:\stuff\seminars\OSGi_workshop\workspace\org.osgi.work

Project Settings

[✓] Create a Java project

Source folder: src

Output folder: bin

Target Platform

This plug-in is targeted to run with:

● Eclipse version: 3.5 or greater

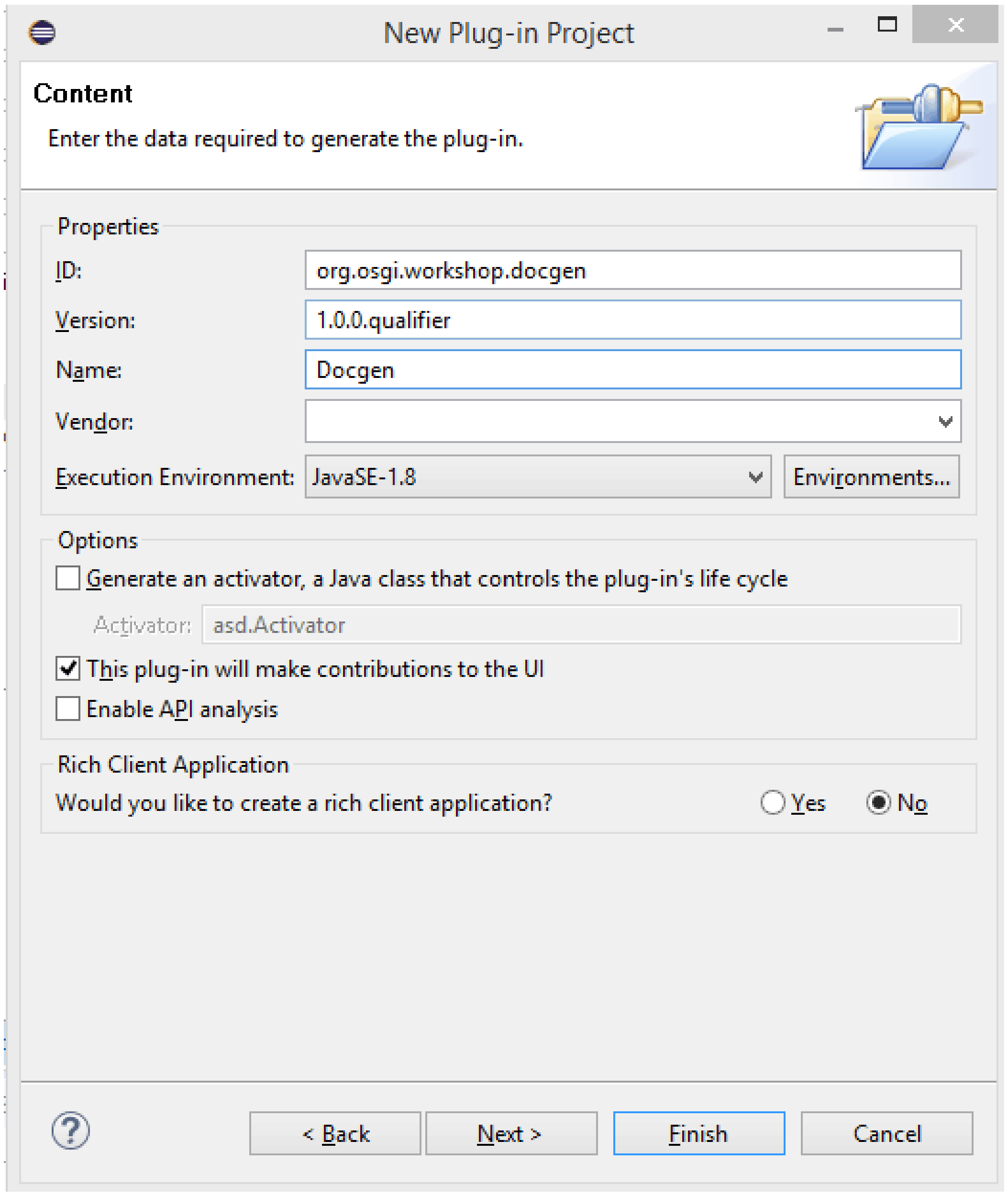
○ an OSGi framework: Equinox

Working sets

[ ] Add project to working sets

Working sets:

Click on **Next** and leave as specified (make sure Execution environment is set to point to JavaSE-1.8) and then click on **Finish**:


New Plug-in Project

Content

Enter the data required to generate the plug-in.

Properties

ID:	org.osgi.workshop.docgen
Version:	1.0.0.qualifier
Name:	Docgen
Vendor:	
Execution Environment:	JavaSE-1.8

Options

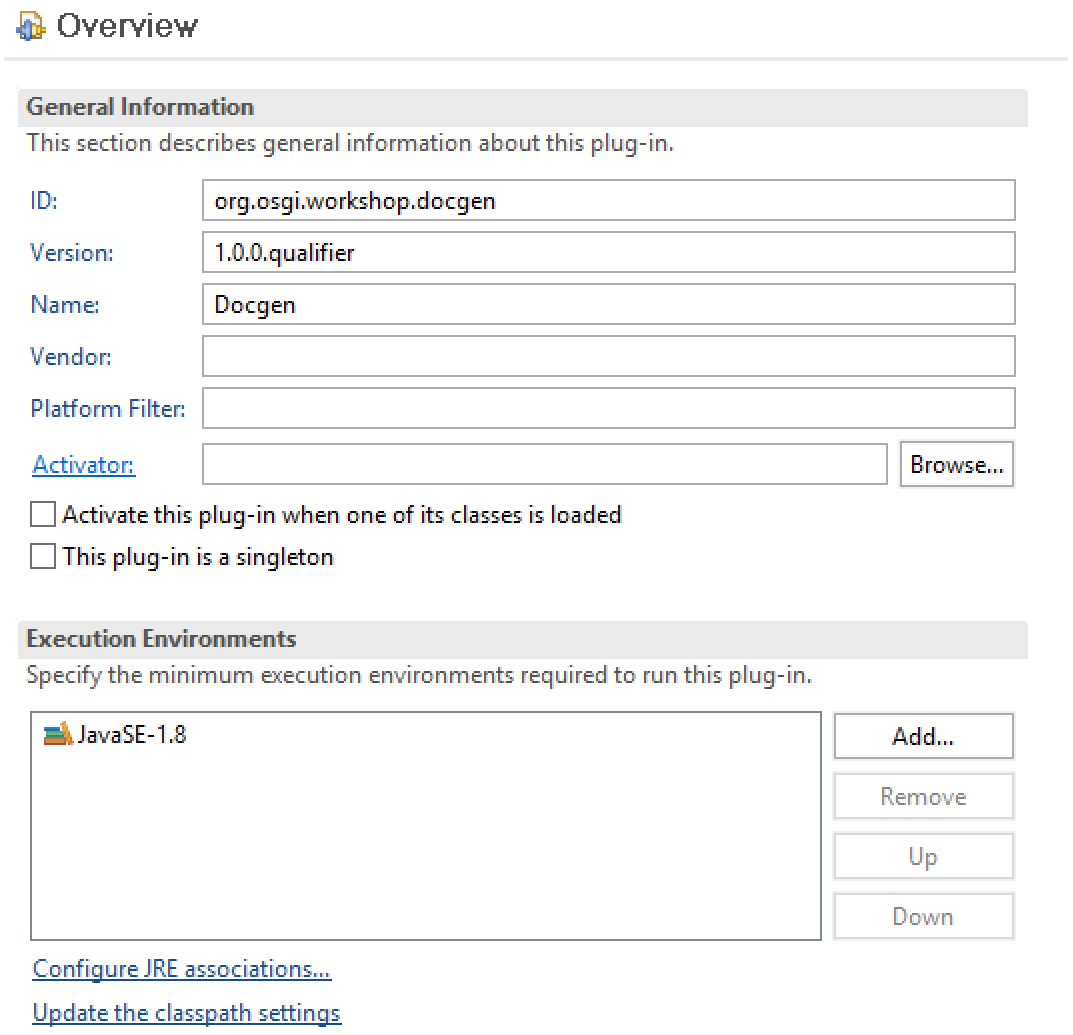
- [ ] Generate an activator, a Java class that controls the plug-in's life cycle
  - Activator: asd.Activator
- [x] This plug-in will make contributions to the UI
- [ ] Enable API analysis

Rich Client Application

Would you like to create a rich client application? Yes [✓ No]

Eclipse generates a plug-in project (OSGi bundle) with a default **MANIFEST.MF** file that can also be edited through the PDE editor.

Next create an activator class that provides the lifecycle methods for our bundle under the **org.osgi.workshop.docgen** package. You can easily through that when you double click on MANIFEST.MF to open the plug-in editor and then click on **Activator** on the **Overview** screen. For the name specify DocgenActivator and click on **Finish**:



Overview

General Information

This section describes general information about this plug-in.

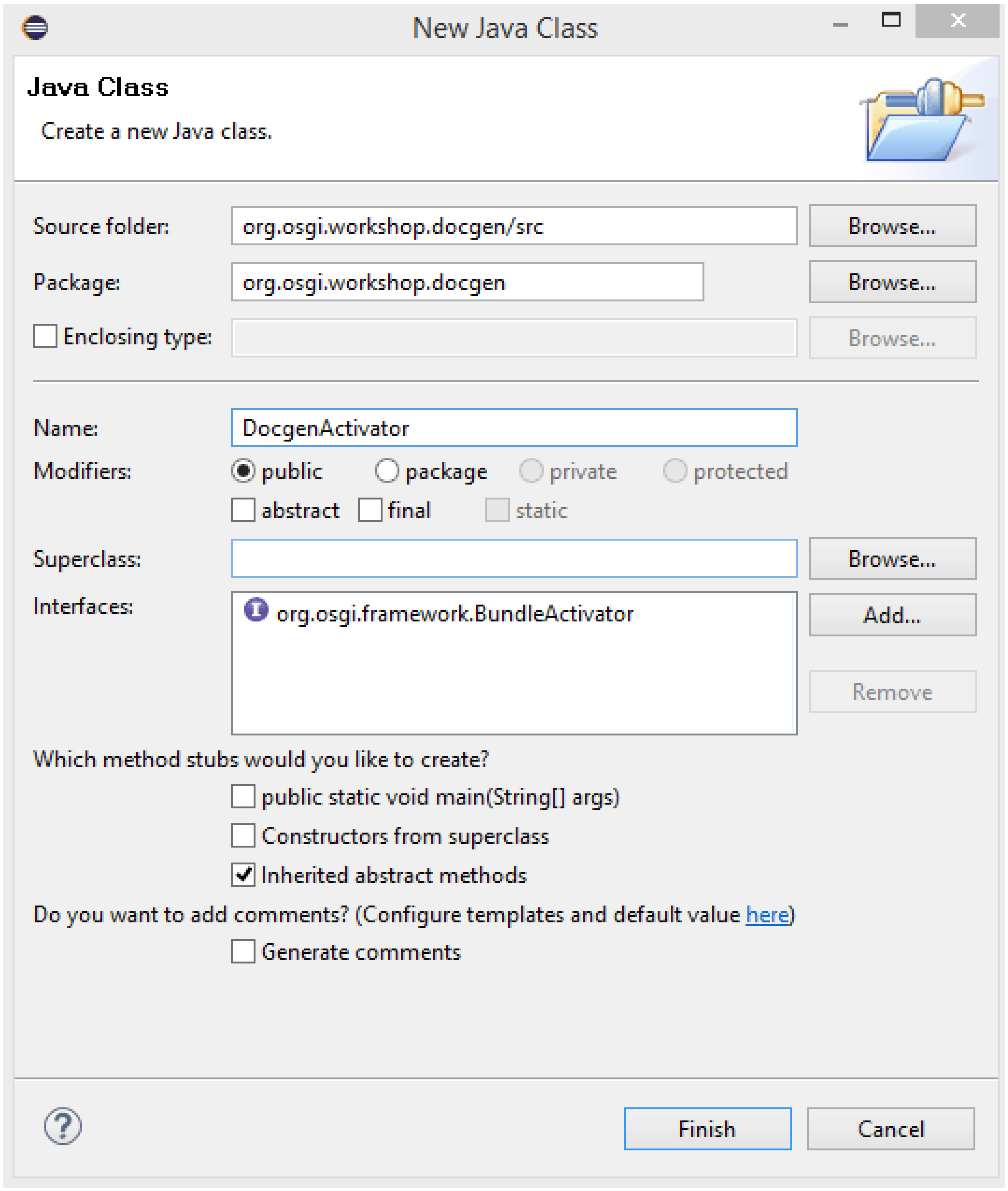
ID:	org.osgi.workshop.docgen
Version:	1.0.0.qualifier
Name:	Docgen
Vendor:	
Platform Filter:	
Activator:	

- [ ] Activate this plug-in when one of its classes is loaded
- [ ] This plug-in is a singleton

Execution Environments

Specify the minimum execution environments required to run this plug-in.

JavaSE-1.8



Java Class

Create a new Java class.

Source folder: org.osgi.workshop.docgen/src [Browse...]

Package: org.osgi.workshop.docgen [Browse...]

☐ Enclosing type: [Browse...]

Name: DocgenActivator

Modifiers: ● public ○ package ○ private ○ protected

☐ abstract ☐ final ☐ static

Superclass: [Browse...]

Interfaces: org.osgi.framework.BundleActivator [Add...] [Remove]

Which method stubs would you like to create?

☐ public static void main(String[] args)

☐ Constructors from superclass

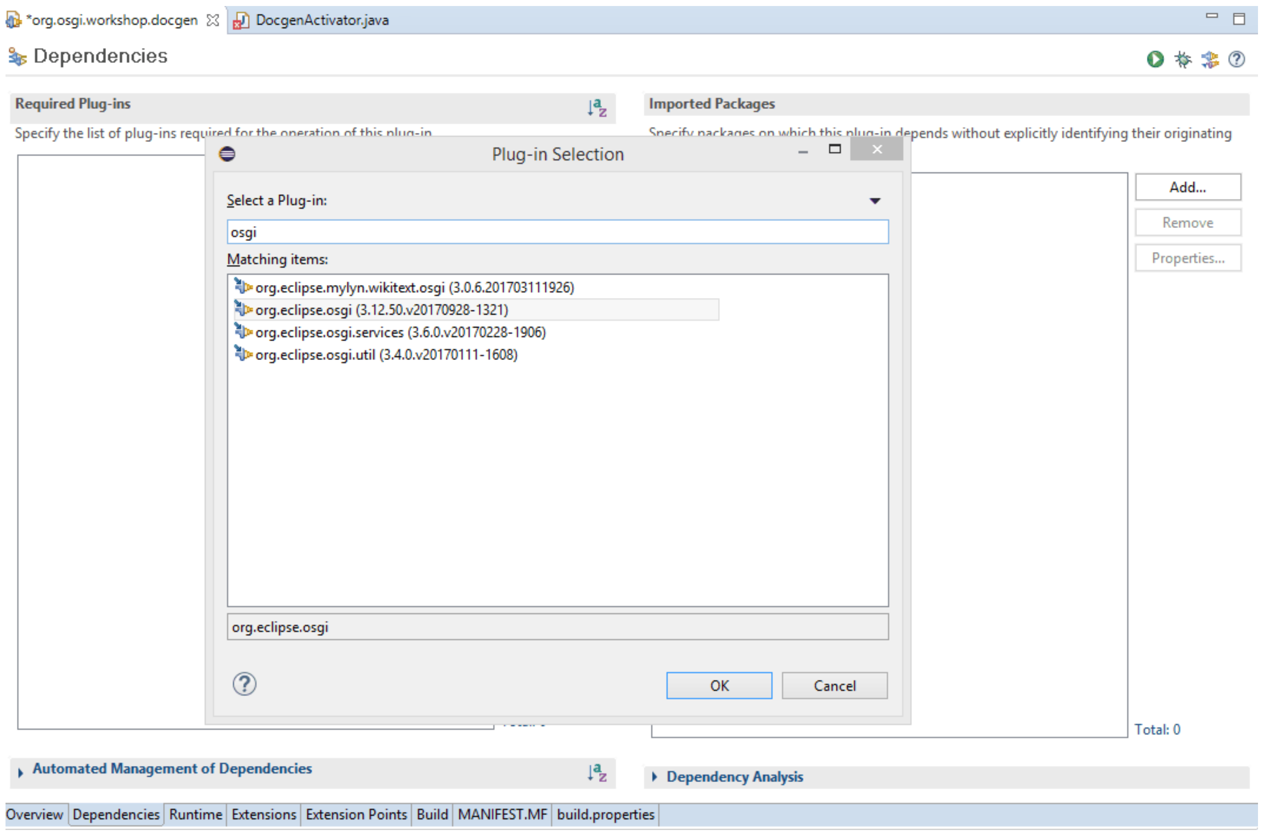
☑ Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)

☐ Generate comments

[Finish] [Cancel]

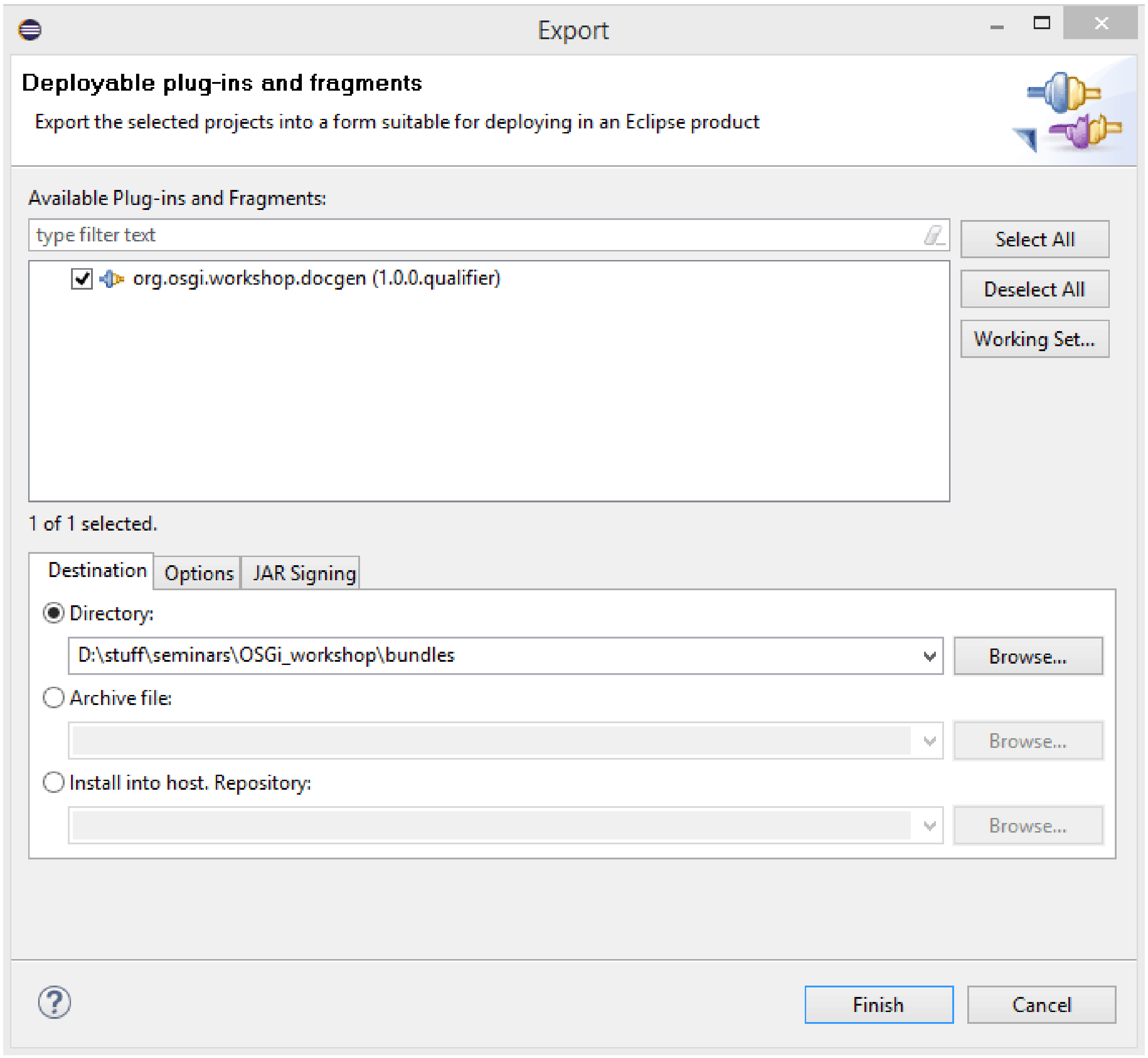
You can see that activator class imports the org.osgi.framework.BundleActivator interface. The interface comes from the **org.eclipse.osgi** bundle that you need to add as a dependency through the **Dependencies** tab:


*org.osgi.workshop.docgen
DocgenActivator.java
Dependencies
Required Plug-ins
Specify the list of plug-ins required for the operation of this plug-in.
Imported Packages
Specify packages on which this plug-in depends without explicitly identifying their originating
Plug-in Selection
Select a Plug-in:
osgi
Matching items:
org.eclipse.mylyn.wikitext.osgi (3.0.6.201703111926)
org.eclipse.osgi (3.12.50.v20170928-1321)
org.eclipse.osgi.services (3.6.0.v20170228-1906)
org.eclipse.osgi.util (3.4.0.v20170111-1608)
org.eclipse.osgi
OK
Cancel
Add...
Remove
Properties...
Total: 0
Automated Management of Dependencies
Dependency Analysis
Overview
Dependencies
Runtime
Extensions
Extension Points
Build
MANIFEST.MF
build.properties

You also need to implement the **start** and **stop** methods from the activator interface. They will be triggered when you start or stop your bundle within the OSGi runtime:

```java
package org.osgi.workshop.docgen;

import java.util.logging.Logger;

import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;

public class DocgenActivator implements BundleActivator {

	private static final Logger LOGGER =
Logger.getLogger(DocgenActivator.class.getName());

	@Override
	public void start(BundleContext arg0) throws Exception {
		LOGGER.info("Starting the Docgen interface bundle ... ");
	}

	@Override
	public void stop(BundleContext arg0) throws Exception {
		LOGGER.info("Stopping the Docgen interface bundle ... ");
	}

}
```

Now we can export our bundle, deploy and run within an OSGi environment. Open again the **Overview** page of the PDE editor and click on **Export Wizard** and specify a proper directory for the creation of the bundle's JAR file and click **Finish**:


Export

Deployable plug-ins and fragments

Export the selected projects into a form suitable for deploying in an Eclipse product

Available Plug-ins and Fragments:

type filter text

[✓] org.osgi.workshop.docgen (1.0.0.qualifier)

1 of 1 selected.

Destination | Options | JAR Signing

(●) Directory: D:\stuff\seminars\OSGi_workshop\bundles

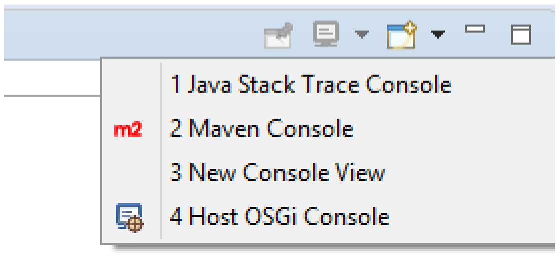
( ) Archive file:

( ) Install into host. Repository:

Equinox

As Eclipse IDE is based on the Equinox OSGi runtime we can open up the OSGi Console within Eclipse and deploy the bundle through it. Pretty much every OSGi runtime implementation comes with an OSGi console that can be used to manage the runtime.

Open the **Console** view within Eclipse and from the upper right corner dropdown click on **Host OSGi Console**:



From within the console you can type the **help** command to see a list of available options. To see a list of bundles deployed within the runtime along with their status you can run the following:

```
ss
```

```
611     RESOLVED    org.bndtools.embeddedrepo_4.2.0.201811240202
                    Master=601
612     ACTIVE      org.bndtools.headless.build.manager_4.2.0.201811240202
613     ACTIVE      org.bndtools.headless.build.plugin.ant_4.2.0.201811240202
614     ACTIVE      org.bndtools.headless.build.plugin.gradle_4.2.0.201811240202
615     STARTING    org.bndtools.templating_4.2.0.201811240202
616     STARTING    org.bndtools.templating.gitrepo_4.2.0.201811240202
617     ACTIVE      org.bndtools.versioncontrol.ignores.manager_4.2.0.201811240202
618     ACTIVE      org.bndtools.versioncontrol.ignores.plugin.git_4.2.0.201811240202
```

Now to install our Docgen bundle we need to use the **install** command and point it to our JAR file on the local file system as follows:

```
install
file:///D:/stuff/seminars/OSGi_workshop/bundles/plugins/org.osgi.workshop.docgen_1.0.0.2018112514
19.jar
```

Once installed you can view plug-in information with the following command:

```
ss org.osgi.workshop.docgen
```

You can see that the bundle is in status **INSTALLED**. Now we need to start it in order to move it to an **ACTIVE** state as follows:

```
start org.osgi.workshop.docgen
```

To stop it you can run the following:

```
stop org.osgi.workshop.docgen
```

You can see that after stopping the bundle it is in status **RESOLVED**.

To uninstall the bundle from the runtime you can run the following:

```
uninstall org.osgi.workshop.docgen
```

Note that you can also run the Equinox OSGi console from outside of Eclipse by navigating to the Eclipse plug-ins directory and running the following:

```
 java -jar org.eclipse.osgi_3.2.0.jar -console
```

Felix

Now we will deploy the bundle within the Felix runtime. Assuming Felix is installed at a proper location navigate to the installation directory and run the following:

```
cd D:\software\felix-framework-6.0.1

java –jar bin/felix.jar
```

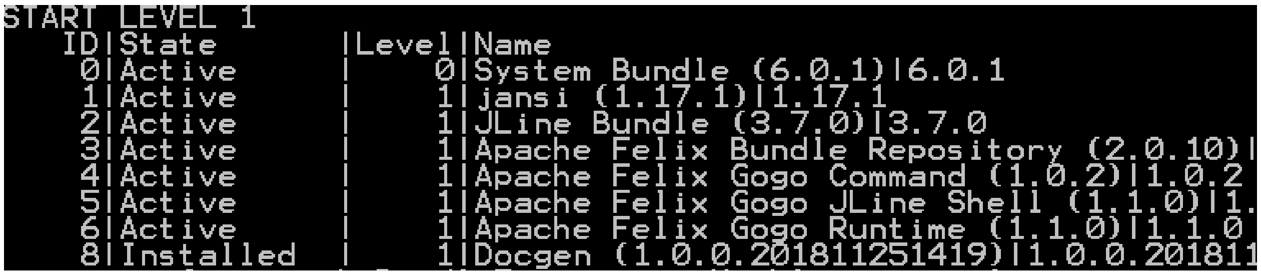
To install the bundle you run exactly the same command as with the Equinox runtime:

```
install
file:///D:/stuff/seminars/OSGi_workshop/bundles/plugins/org.osgi.workshop.docgen_1.0.0.2018112514
19.jar
```

In order to list installed bundles you can run the following:

```
lb
```

You can see that the org.osgi.workshop.docgen bundle is installed:



```
START LEVEL 1
   ID|State      |Level|Name
    0|Active     |    0|System Bundle (6.0.1)|6.0.1
    1|Active     |    1|jansi (1.17.1)|1.17.1
    2|Active     |    1|JLine Bundle (3.7.0)|3.7.0
    3|Active     |    1|Apache Felix Bundle Repository (2.0.10)|
    4|Active     |    1|Apache Felix Gogo Command (1.0.2)|1.0.2
    5|Active     |    1|Apache Felix Gogo JLine Shell (1.1.0)|1.
    6|Active     |    1|Apache Felix Gogo Runtime (1.1.0)|1.1.0
    8|Installed  |    1|Docgen (1.0.0.201811251419)|1.0.0.201811
```

However if you try to start the bundle by running the following:

```
start 8
```

You will get the following error:

```
org.osgi.framework.BundleException: Unable to resolve org.osgi.workshop.docgen [
8](R 8.0): missing requirement [org.osgi.workshop.docgen [8](R 8.0)] osgi.wiring
.bundle; (&(osgi.wiring.bundle=org.eclipse.osgi)(bundle-version>=3.12.50)) Unres
olved requirements: [[org.osgi.workshop.docgen [8](R 8.0)] osgi.wiring.bundle; (
&(osgi.wiring.bundle=org.eclipse.osgi)(bundle-version>=3.12.50))]
```

The reason is in a way obvious: There is not OSGi deployed in the Felix runtime named **org.eclipse.osgi**. In fact our bundle requires just the **org.osgi.framework** disregarding which bundle is providing it. This is an essential feature of the OSGi runtime – you can require particular packages along with a particular version range disregarding which bundle in the runtime is providing it. In the next section we will dive into more details on importing and exporting packages.
To resolve the particular issue we need to open the MANIFEST.MF file, remove the Require-Bundle section and add the following import the **org.osgi.framework** package**:**

```
Import-Package: org.osgi.framework
```

Then you need to re-export the bundle and try to install it again within the Felix runtime and this time you would be able to start the bundle using the **start** command and the ID of the bundle.

Bundle development

Now we will extend our bundle with particular capabilities for document generation and add two more bundles that provide services for the generation of PDF and Word documents.

Importing/exporting packages

Create the **org.osgi.workshop.docgen.api** package and inside the package the create the DocumentGenerator interface as follows:

```
package org.osgi.workshop.docgen.api;

public interface DocumentGenerator {

	public void generate(String filename, String text);

}
```

As you can see the interface is pretty simple and provides a common API for all document generators implemented in separate bundles. Export the **org.osgi.workshop.docgen.api** package from the bundle by adding the following Export-Package attribute in the MANIFEST.MF file of the bundle (you can also do that from the PDE manifest editor):

```
Export-Package: org.osgi.workshop.docgen.api
```

Now create another bundle called **org.osgi.workshop.docgen.pdf** along with a bundle activator provided by the **org.osgi.workshop.docgen.pdf.Activator** class.

Create another package called **org.osgi.workshop.docgen.pdf.services** that will provide the various service implementations from the PDF generator bundle. Next import the **org.osgi.framework** package required by the bundle activator and the **org.osgi.workshop.docgen.api** package from the **org.osgi.workshop.docgen:**

```
Import-Package: org.osgi.framework;version="1.8.0",
 org.osgi.workshop.docgen.api
```

Next we are going to create a concrete implementation of the **org.osgi.workshop.docgen.api.DocumentGenerator** service but first we need to include a proper library that is going to handle the actual PDF generation. Popular choices of such libraries are iText and PdfBox. We are going to use iText as it references less dependencies and hence is easier to use for the purpose of demonstration. The first approach that comes to mind is to download and include the library on the bundle classpath along with it dependencies. Download the JAR files of the library and its dependencies from the following URLs and put them under the **lib** folder in the bundle:

http://central.maven.org/maven2/com/itextpdf/itextpdf/5.0.6/itextpdf-5.0.6.jar

http://central.maven.org/maven2/bouncycastle/bcmail-jdk14/138/bcmail-jdk14-138.jar

http://central.maven.org/maven2/bouncycastle/bcprov-jdk14/138/bcprov-jdk14-138.jar

http://central.maven.org/maven2/org/bouncycastle/bctsp-jdk14/1.38/bctsp-jdk14-1.38.jar

Add the following entries to the Bundle-ClassPath:

```
Bundle-ClassPath: lib/bcmail-jdk14-138.jar,
 lib/bcprov-jdk14-138.jar,
 lib/bctsp-jdk14-1.38.jar,
 lib/itextpdf-5.0.6.jar,
 .
```

In the **org.osgi.workshop.docgen.pdf.services** create the **PdfDocumentGenerator** interface as follows:

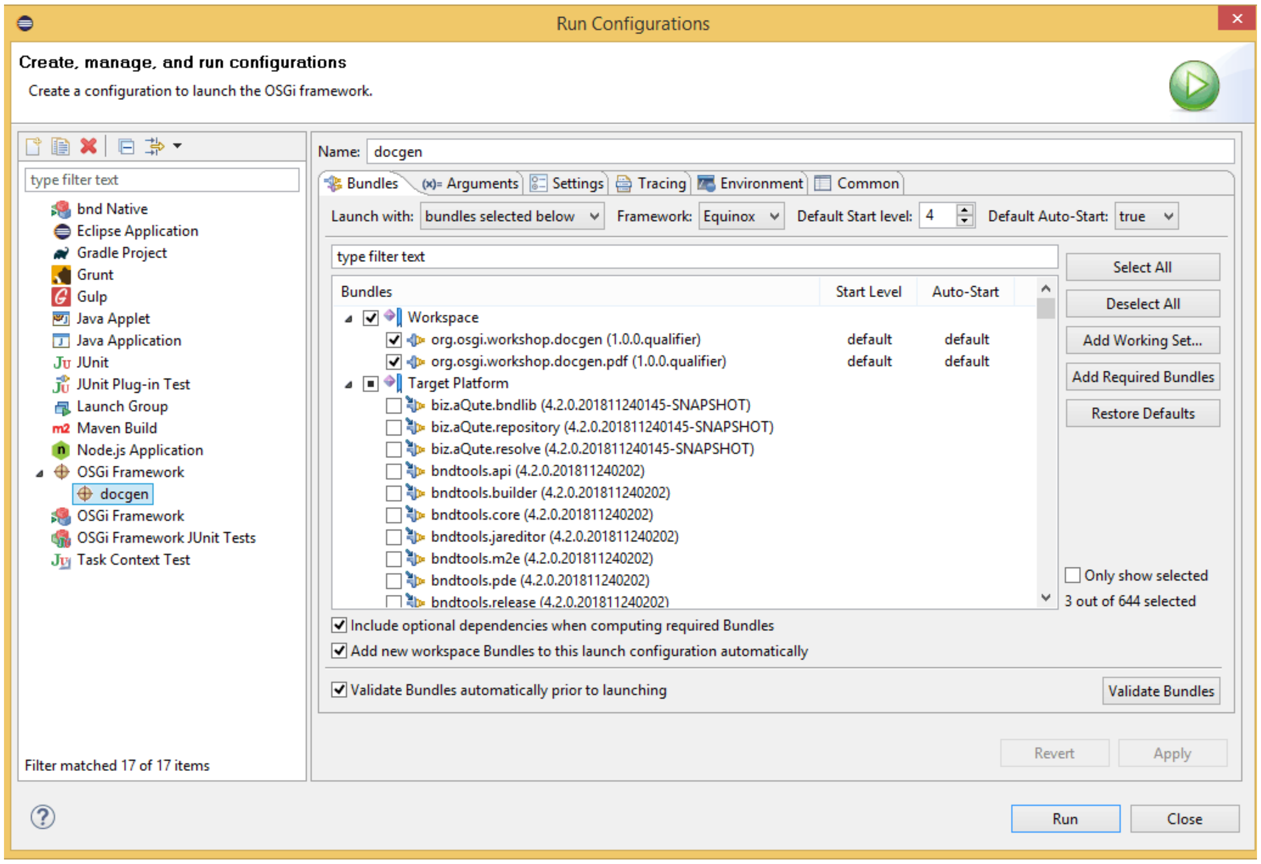
```
package org.osgi.workshop.docgen.pdf.services;

import java.io.FileOutputStream;

import org.osgi.workshop.docgen.api.DocumentGenerator;

import com.itextpdf.text.Document;
import com.itextpdf.text.Paragraph;
import com.itextpdf.text.pdf.PdfWriter;

public class PdfDocumentGenerator implements DocumentGenerator {

	@Override
	public void generate(String filename, String text) {
		try {
```

```
                  Document document = new Document();
                  PdfWriter.getInstance(document, new
                        FileOutputStream(filename));
                  document.open();
                  document.add(new Paragraph(text));
                  document.close();
            } catch (Exception ex) {
                  ex.printStackTrace();
            }
      }
}
```

To demonstrate the usage of the above generator include the following test code in the start() method of the bundle's activator:

```
new PdfDocumentGenerator().generate("D:\\test.pdf", "Hello OSGi");
```

Now instead of installing the API and PDF generator bundles in either the OSGi or Felix consoles you can create a new OSGi run configuration through eclipse from **Run** -> **Run Configurations** and specifying a new configuration under the **OSGi Framework** menu with only the **org.eclipse.osgi** bundle specified in the Target Platform as follows:


Run Configurations

Create, manage, and run configurations

Create a configuration to launch the OSGi framework.

type filter text

- bnd Native
- Eclipse Application
- Gradle Project
- Grunt
- Gulp
- Java Applet
- Java Application
- JUnit
- JUnit Plug-in Test
- Launch Group
- Maven Build
- Node.js Application
- OSGi Framework
  - docgen
- OSGi Framework
- OSGi Framework JUnit Tests
- Task Context Test

Filter matched 17 of 17 items

Name: docgen

Bundles | Arguments | Settings | Tracing | Environment | Common

Launch with: bundles selected below   Framework: Equinox   Default Start level: 4   Default Auto-Start: true

type filter text

Bundles	Start Level	Auto-Start
☑ Workspace		
☑ org.osgi.workshop.docgen (1.0.0.qualifier)	default	default
☑ org.osgi.workshop.docgen.pdf (1.0.0.qualifier)	default	default
▣ Target Platform		
☐ biz.aQute.bndlib (4.2.0.201811240145-SNAPSHOT)		
☐ biz.aQute.repository (4.2.0.201811240145-SNAPSHOT)		
☐ biz.aQute.resolve (4.2.0.201811240145-SNAPSHOT)		
☐ bndtools.api (4.2.0.201811240202)		
☐ bndtools.builder (4.2.0.201811240202)		
☐ bndtools.core (4.2.0.201811240202)		
☐ bndtools.jareditor (4.2.0.201811240202)		
☐ bndtools.m2e (4.2.0.201811240202)		
☐ bndtools.pde (4.2.0.201811240202)		
☐ bndtools.release (4.2.0.201811240202)		

Select All | Deselect All | Add Working Set... | Add Required Bundles | Restore Defaults

☐ Only show selected

3 out of 644 selected

☑ Include optional dependencies when computing required Bundles

☑ Add new workspace Bundles to this launch configuration automatically

☑ Validate Bundles automatically prior to launching   Validate Bundles

Revert | Apply

Run | Close

Observe that a **test.pdf** file is created on D: drive.

We include the third party libraries to the bundle classpath but a better, more OSGi-like manner to use these libraries is to install them as bundles in the OSGi runtime. Many third-party libraries come with OSGi metadata (proper MANIFEST.MF file). There are some good sources of "OSGified" libraries such as the Apache ServiceMix project. The four libraries we used for the PDF generator project don't have proper OSGi metadata so the options we have in order to use them as OSGi bundles are:

- to find versions of the libraries that have the proper OSGi metadata;
- to generate the proper OSGi metadata and patch the JAR files of the libraries – the Bnd tool can be used for the purpose.

Declaring and consuming services

The next thing we will do is to register our PDF document generator as a service implementation of the document generator service. To do that we need to enhance the the start() method of the activator as follows (comment out the line that creates an instance of the PDF document generator that we added in the previous section):

```
	@Override
	public void start(BundleContext context) throws Exception {
		// new PdfDocumentGenerator().generate("D:\\test.pdf", "Hello
				OSGi");
		 context.registerService(DocumentGenerator.class.getName(), new
				PdfDocumentGenerator(), null);
	}
```

Run again the API and PDF generator bundles (in the OSGi Run Configuration that is already created click the **Add Required Bundle** window in order to include the OSGi console and dependent bundles) and through the OSGi console type the following and observe if the service implementation is registered properly:

```
bundle org.osgi.workshop.docgen.pdf
```

You should observe an output similar to the following that indicates the registered service:

```
org.osgi.workshop.docgen.pdf_1.0.0.qualifier [2]
  Id=2, Status=ACTIVE      Data
Root=D:\stuff\seminars\OSGi_workshop\workspace\.metadata\.plugins\org.eclipse.
pde.core\docgen\org.eclipse.osgi\2\data
  "Registered Services"
    {org.osgi.workshop.docgen.api.DocumentGenerator}={service.id=32,
service.bundleid=2, service.scope=singleton}
  No services in use.
  No exported packages
  Imported packages
    org.osgi.framework; version="1.8.0" <org.eclipse.osgi_3.12.50.v20170928-
1321 [0]>
```

```
    org.osgi.workshop.docgen.api; version="0.0.0"
<org.osgi.workshop.docgen_1.0.0.qualifier [3]>
  No fragment bundles
  No required bundles
```

In order to demonstrate how to use the registered service we will create another bundle called **org.osgi.workshop.docgen.demo** with an activator class named Activator under the **org.osgi.workshop.docgen.demo** package. Also in the bundle's metadata add the **org.osgi.workshop.docgen.api** package as imported. In the start() method of the demo bundle's activator add the following code to retrieve and use the service to generate a document:

```
ServiceReference<DocumentGenerator> serviceRef =
      context.getServiceReference(DocumentGenerator.class);
if(serviceRef != null) {
      DocumentGenerator service = (DocumentGenerator) context.
                getService(serviceRef);
      service.generate("D:\\test2.pdf", "Another test document.");
}
```

Creating a service listener

In the demo bundle that we created we have a general problem: the service we trying to use might not be existing. If for some reason the PDF generator bundle is deployed after the demo bundle the previous scenario will not be working as we expect. The OSGi framework provides several ways to deal with this problem: using a service listener, a service tracker utility, declarative services or OSGi blueprint. We will look into the first three approaches.

Create a new package called **org.osgi.workshop.docgen.demo.listeners** with a class called **DocumentGeneratorServiceListener** that has the following:

```
package org.osgi.workshop.docgen.demo.listeners;

import org.osgi.framework.ServiceEvent;
import org.osgi.framework.ServiceListener;
import org.osgi.workshop.docgen.api.DocumentGenerator;

public class DocumentGeneratorServiceListener implements ServiceListener {

      @Override
      public void serviceChanged(ServiceEvent event) {
            if (ServiceEvent.REGISTERED == event.getType()) {
                  DocumentGenerator docGen = (DocumentGenerator)
event.getServiceReference().getBundle().getBundleContext()
                              .getService(event.getServiceReference());
```

```
                        docGen.generate("D:\\test3.pdf", "Yet aother sample
service.");
                }
        }
}
```

Also comment out the code from the start() method of the activator and add the following code that registers the service listener instead:

```
                context.addServiceListener(new DocumentGeneratorServiceListener(),
                        "(" + Constants.OBJECTCLASS + "=" +
                 DocumentGenerator.class.getName() + ") ");
```

The first argument is the listener instance and the second is a filter (format of filters being defined in the OSGi spec) that specifies that the listener is going to receive events only if the listener interface corresponds to the one provided. We might also add more listener interfaces separated by the | operator. There is also an **addServiceListener** method that accepts just the listener instance but it will receive events for all services being modified in the service registry. If you run all the bundles at once from the Eclipse run configuration most likely you will not see a test3.pdf generated because the PDF bundle is registered first and the listener does not receive any event. To trigger an event you can stop and then start again the PDF bundle as follows (and observe that the test3.pdf is generated):

```
stop org.osgi.workshop.docgen.pdf
start org.osgi.workshop.docgen.pdf
```

The serviceChanged() method is called every time the definition of a services changes. The **org.osgi.framework.ServiceListener** API does not provide methods to indicate when a service has been registered or unregistered – only when it has been modified. For that reason the service has the drawback that we need to make an extra check in order to determine if the service is registered, unregistered or modified. In the above example we only call the generate() method when the service is registered. Yet we have the drawback that if the service is already registered the document won't be generated.

Tracking services

To solve the issue with the service listener we can use another utility called a service tracker. A service tracker must implement the **org.osgi.util.tracker.ServiceTrackerCustomizer** interface. Add the **org.osgi.util.tracker** package to the imported packages of the demo bundle. Create the following class under the **org.osgi.workshop.docgen.demo.tracker** package:

```
package org.osgi.workshop.docgen.demo.trackers;

import org.osgi.framework.BundleContext;
import org.osgi.framework.ServiceReference;
import org.osgi.util.tracker.ServiceTrackerCustomizer;
import org.osgi.workshop.docgen.api.DocumentGenerator;

public class DocumentGeneratorServiceTracker implements
ServiceTrackerCustomizer<DocumentGenerator, Object> {

	private BundleContext context;

	public DocumentGeneratorServiceTracker(BundleContext context) {
		this.context = context;
	}

	@Override
	public Object addingService(ServiceReference<DocumentGenerator>
serviceRef) {
		DocumentGenerator docGen = context.getService(serviceRef);
		docGen.generate("D:\\test4.pdf", "Hello, ServiceTracker.");
		return docGen;
	}

	@Override
	public void modifiedService(ServiceReference<DocumentGenerator>
serviceRef, Object serviceImpl) {
	}

	@Override
	public void removedService(ServiceReference<DocumentGenerator>
serviceRef, Object serviceImpl) {
	}

}
```

Now comment out the registration of the service listener in the start() method of the activator and instead register the created service tracker as follows:

```
		DocumentGeneratorServiceTracker trackerCustomizer = new
DocumentGeneratorServiceTracker(
				context);
		serviceTracker = new ServiceTracker(context, DocumentGenerator.class
				.getName(), trackerCustomizer);
		serviceTracker.open();
```

You also need to declare an instance field that keeps the service tracker:

```
private ServiceTracker serviceTracker;
```

You also need to close the tracker when the demo bundle is stopped so add the following code to the stop() method:

```
serviceTracker.close();
```

Now run again the three bundles and observe that the test4.pdf is created.

Creating declarative services using XML configuration

Declarative services are a dependency injection mechanism that allows us to manage OSGi service using a XML-based DSL that additionally provides dependency injection mechanism for service injection and eliminated the need to a service listener or a service tracker. We are now going to convert the code that handles our document generator service to declarative service.

First we need to comment out the service registration from the start() method of PDF document generator's activator. Then add the OSGI-INF/service.xml file with the following definition:

```
<?xml version="1.0" encoding="UTF-8"?>
<scr:component xmlns:scr="http://www.osgi.org/xmlns/scr/v1.1.0"
name="org.osgi.workshop.docgen.api.DocumentGenerator">
   <implementation
class="org.osgi.workshop.docgen.pdf.services.PdfDocumentGenerator"/>
   <service>
      <provide interface="org.osgi.workshop.docgen.api.DocumentGenerator"/>
   </service>
</scr:component>
```

Also make sure that the OSGI-INF folder is added to the build.properties so that it is included in the bundle distribution:

```
source.. = src/
output.. = bin/
bin.includes = META-INF/,\
               .,\
               lib/bcmail-jdk14-138.jar,\
               lib/bcprov-jdk14-138.jar,\
               lib/bctsp-jdk14-1.38.jar,\
               lib/itextpdf-5.0.6.jar,\
               OSGI-INF/
```

And also register the service XML file in the bundle MANIFEST as follows:

```
Bundle-ActivationPolicy: lazy
Service-Component: OSGI-INF/service.xml
```

Also include the **org.eclipse.equinox.ds** and **org.apache.felix.scr** bundles to the target runtime of your run configuration before running the bundles. Run the bundles and observe that the test4.pdf file is generated.

Next we are going to remove the service tracker in favor of using declarative services. Comment out the code in the start() and stop() methods of the demo bundle that create, open and close the service tracker we created.

Create the OSGI-INF/service.xml file with the following definition:

```
<?xml version="1.0" encoding="UTF-8"?>
<scr:component xmlns:scr="http://www.osgi.org/xmlns/scr/v1.1.0"
name="activator">
   <implementation class="org.osgi.workshop.docgen.demo.Activator"/>
   <reference bind="bind"
                          cardinality="1..1"

      interface="org.osgi.workshop.docgen.api.DocumentGenerator"
                          name="org.osgi.workshop.docgen.api.DocumentGenerator"
                          policy="static"
                          unbind="unbind"/>
</scr:component>
```

Again make sure that the OSGI-INF folder is added to the build.properties so that it is included in the bundle distribution:

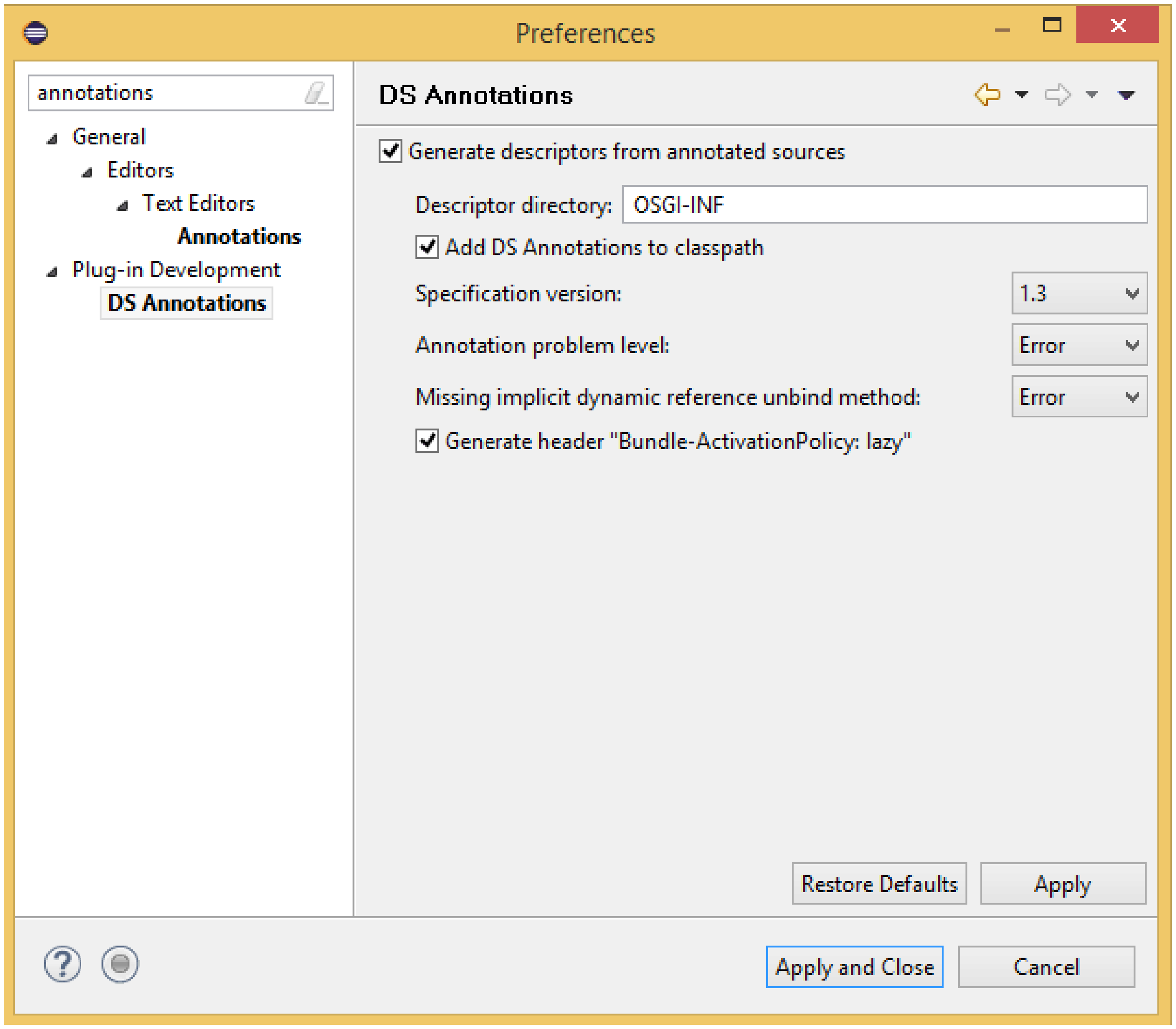
```
source.. = src/
output.. = bin/
bin.includes = META-INF/,\
               .,\
               lib/bcmail-jdk14-138.jar,\
               lib/bcprov-jdk14-138.jar,\
               lib/bctsp-jdk14-1.38.jar,\
               lib/itextpdf-5.0.6.jar,\
               OSGI-INF/
```

And also register the service XML file in the bundle MANIFEST as follows:

```
Bundle-ActivationPolicy: lazy
Service-Component: OSGI-INF/service.xml
```

Creating declarative services using annotations

To simplify the usage of declarative services the OSGi DS specification provides a set of build time annotations out of which the proper declarative service XML files can be generated by means of proper IDE support. Eclipse PDE already provides a builder for declarative service from annotations that must be enabled by going to **Preferences** (or Project Properties) -> **Plug-in Development** -> **DS Annotations** and checking the **Generate descriptors from annotated sources** option. For the service directory leave OSGI-INF as specified by default:


Preferences

annotations

- General
  - Editors
    - Text Editors
      - **Annotations**
- Plug-in Development
  - **DS Annotations**

DS Annotations

☑ Generate descriptors from annotated sources

Descriptor directory: OSGI-INF

☑ Add DS Annotations to classpath

Specification version: 1.3

Annotation problem level: Error

Missing implicit dynamic reference unbind method: Error

☑ Generate header "Bundle-ActivationPolicy: lazy"

Remove the service.xml files from the OSGI-INF directories of the demo and PDF bundles.

Also add the **org.osgi.service.component.annotations** package (latest version) as dependency to the PDF and demo bundles in order to be able to use the DS annotations.

Add the following annotations:

@Component(service=DocumentGenerator.class) on the PDFDocumentGenerator class from the PDF bundle

@Component(service= {}) on the activator of the demo bundle (the {} value for the service specifies that no service implementation should be exposed from the component class)

@Reference(unbind="unbind", cardinality=ReferenceCardinality.MANDATORY, policy=ReferencePolicy.STATIC) to the bind() method of the demo activator class

Observe that after adding the corresponding annotations proper XMLfiles are generated under the OSGI-INF directories of the PDF and demo bundles and also the bundle manifest are modified automatically to point to the generated XML files.

Observe that the test5.pdf file is created.

Bundle build and deployment

For the rest of the examples in the workshop we will switch to Apache Karaf (download link is at the beginning of the workshop). All of the previous examples can be deployed under a Karaf environment. Unzip the Karaf archive to a proper location. Start the karaf container by navigating to the installation directory of Karaf (by default Karaf runs Apache Felix but you can switch to Equinox):

```
bin\karaf.bat
```

Install the Karaf web console by using the following commands:

```
feature:install webconsole
```

To verify that the webconsole is running navigate to http://localhost:8181/system/console (username/password: karaf/karaf).

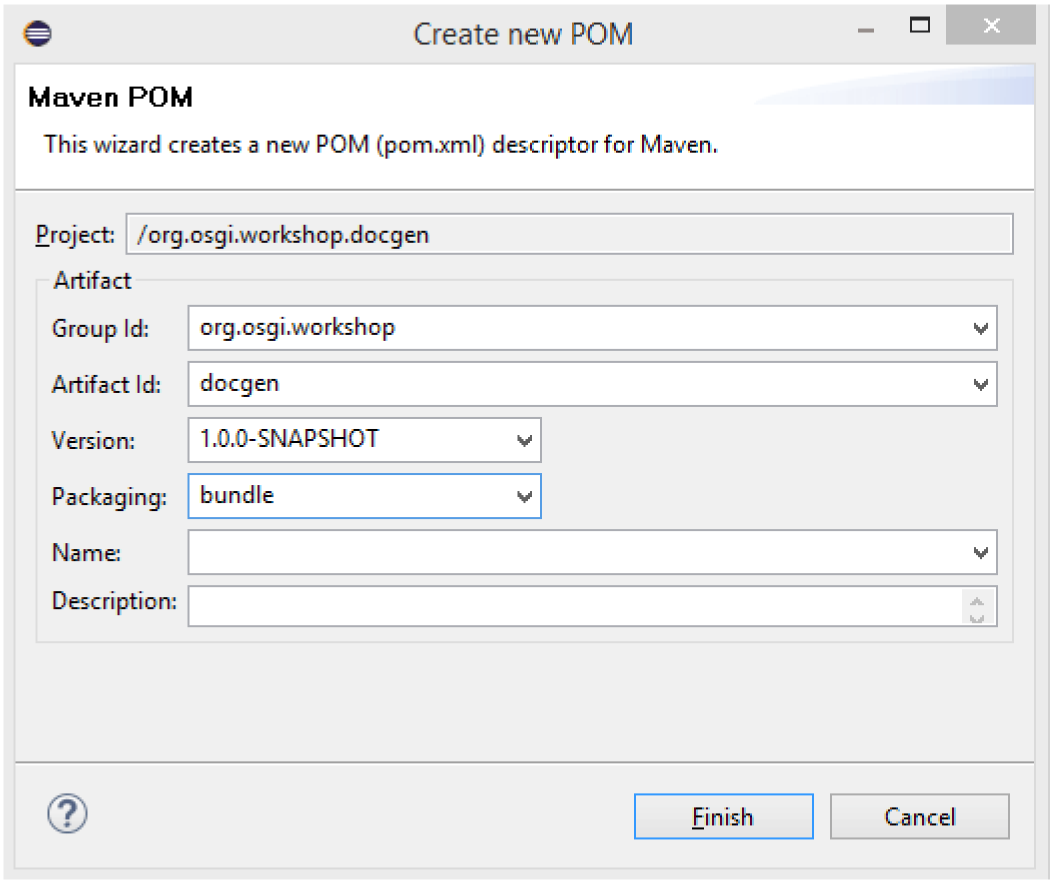
Building OSGi bundles

In this section we will explore some of the most widely used option to build OSGi bundles with Maven. It is typically cumbersome to write and maintain the MANIFEST.MF file by hand and moreover we would typically like to externalize that in our build process and use build variables when defining some of the bundle metadata entities. In particular we will look into four ways of building OSGi bundles using the following Maven plugins:

- maven-bundle-plugin
- bnd-maven-plugin
- karaf-maven-plugin
- tycho-maven-plugin

Maven bundle plug-in

First we are going to convert our projects to Maven projects by right-clicking on each of them and specifying **Configure** -> **Convert to Maven project.** For each of them specify as **groupId org.osgi.workshop** and for the **artifactId** the rest of the full project name. For packaging type specify **bundle**. For example for the docgen API bundle specify the following:



Maven POM

This wizard creates a new POM (pom.xml) descriptor for Maven.

Project: /org.osgi.workshop.docgen

Artifact

Group Id: org.osgi.workshop

Artifact Id: docgen

Version: 1.0.0-SNAPSHOT

Packaging: bundle

Name:

Description:

In the generated pom.xml file add the following configuration:

```
	<properties>
		<maven.compiler.source>1.8</maven.compiler.source>
		<maven.compiler.target>1.8</maven.compiler.target>
	</properties>

	<dependencies>
		<dependency>
			<groupId>org.apache.felix</groupId>
			<artifactId>org.osgi.core</artifactId>
			<version>1.0.0</version>
		</dependency>
	</dependencies>

	<build>
		<sourceDirectory>src</sourceDirectory>
		<plugins>
			<plugin>
				<groupId>org.apache.felix</groupId>
```

```
                        <artifactId>maven-bundle-plugin</artifactId>
                        <version>4.1.0</version>
                        <extensions>true</extensions>
                        <configuration>
                              <instructions>
                                    <_include>META-INF/MANIFEST.MF</_include>
                                    <Export-
Package>org.osgi.workshop.docgen.api</Export-Package>
                              </instructions>
                        </configuration>
                  </plugin>
            </plugins>
      </build>
```

In the above example we add a dependency to the OSGi core implementation bundle provided by Apache Felix and the dependency is used by the maven-bundle-plugin when it tries to compile the bundle. In the configuration section of the plug-in we define the bundle manifest entries that are to be included to the existing MANIFEST.MF file (specified with the **_include** element). If a MANIFEST.MF file does not exist then we can omit the **_include** element and the manifest will be generated. For the PDF and Demo bundles include the same content above without the Export-Package element.

In addition you need to add some extra dependencies to the pom.xml file of the PDF and Demo bundles and some additional BND instructions for the declarative services and bundle classpath. Add the following descriptions for the two bundles:

(for PDF bundle)

```
      <properties>
            <maven.compiler.source>1.8</maven.compiler.source>
            <maven.compiler.target>1.8</maven.compiler.target>
      </properties>

      <dependencies>
            <dependency>
                  <groupId>org.apache.felix</groupId>
                  <artifactId>org.osgi.core</artifactId>
                  <version>1.0.0</version>
                  <scope>provided</scope>
            </dependency>

            <dependency>
                  <groupId>org.osgi.workshop</groupId>
                  <artifactId>docgen</artifactId>
                  <version>1.0.0-SNAPSHOT</version>
                  <scope>provided</scope>
            </dependency>

            <dependency>
                  <groupId>org.osgi</groupId>

      <artifactId>org.osgi.service.component.annotations</artifactId>
                  <version>1.3.0</version>
            </dependency>
            <dependency>
                  <groupId>bcmail</groupId>
                  <artifactId>bcmail</artifactId>
                  <version>1.0.0</version>
                  <scope>system</scope>
                  <systemPath>${project.basedir}\lib\bcmail-jdk14-
138.jar</systemPath>
            </dependency>

            <dependency>
                  <groupId>bcprov</groupId>
                  <artifactId>bcprov</artifactId>
                  <version>1.0.0</version>
                  <scope>system</scope>
                  <systemPath>${project.basedir}\lib\bcprov-jdk14-
138.jar</systemPath>
            </dependency>

            <dependency>
                  <groupId>bctsp</groupId>
                  <artifactId>bctsp</artifactId>
                  <version>1.0.0</version>
                  <scope>system</scope>
                  <systemPath>${project.basedir}\lib\bctsp-jdk14-
1.38.jar</systemPath>
            </dependency>

            <dependency>
                  <groupId>itextpdf</groupId>
                  <artifactId>itextpdf</artifactId>
                  <version>1.0.0</version>
                  <scope>system</scope>
                  <systemPath>${project.basedir}\lib\itextpdf-
5.0.6.jar</systemPath>
            </dependency>

            <dependency>
                  <groupId>org.osgi</groupId>

      <artifactId>org.osgi.service.component.annotations</artifactId>
                  <version>1.3.0</version>
                  <scope>provided</scope>
            </dependency>

      </dependencies>

      <build>
            <sourceDirectory>src</sourceDirectory>
            <plugins>
                  <plugin>
                        <groupId>org.apache.felix</groupId>
                        <artifactId>maven-bundle-plugin</artifactId>
                        <version>4.1.0</version>
                        <extensions>true</extensions>
                        <configuration>
                              <instructions>
```

```
                                    <_include>META-INF/MANIFEST.MF</_include>
                                    <Service-Component>OSGI-INF/*</Service-
Component>
                                    <Bundle-ClassPath>lib/bcmail-jdk14-
138.jar,lib/bcprov-jdk14-138.jar,lib/bctsp-jdk14-1.38.jar,lib/itextpdf-
5.0.6.jar,.</Bundle-ClassPath>
                                    <Embed-Directory>/lib</Embed-Directory>
                                    <Embed-Dependency>*;scope=system</Embed-
Dependency>
                              </instructions>
                        </configuration>
                  </plugin>
            </plugins>
      </build>
```

(For demo bundle)

```
      <properties>
            <maven.compiler.source>1.8</maven.compiler.source>
            <maven.compiler.target>1.8</maven.compiler.target>
      </properties>

      <dependencies>
            <dependency>
                  <groupId>org.apache.felix</groupId>
                  <artifactId>org.osgi.core</artifactId>
                  <version>1.0.0</version>
                  <scope>provided</scope>
            </dependency>

            <dependency>
                  <groupId>org.osgi</groupId>

      <artifactId>org.osgi.service.component.annotations</artifactId>
                  <version>1.3.0</version>
                  <scope>provided</scope>
            </dependency>

            <dependency>
                  <groupId>org.osgi</groupId>
                  <artifactId>org.osgi.util.tracker</artifactId>
                  <version>1.5.2</version>
                  <scope>provided</scope>
            </dependency>

            <dependency>
                  <groupId>org.apache.felix</groupId>
                  <artifactId>org.apache.felix.framework</artifactId>
                  <version>6.0.0</version>
            </dependency>

            <dependency>
                  <groupId>org.osgi.workshop</groupId>
                  <artifactId>docgen</artifactId>
                  <version>1.0.0-SNAPSHOT</version>
                  <scope>provided</scope>
            </dependency>

      </dependencies>

      <build>
            <sourceDirectory>src</sourceDirectory>
            <plugins>
                  <plugin>
                        <groupId>org.apache.felix</groupId>
                        <artifactId>maven-bundle-plugin</artifactId>
                        <version>4.1.0</version>
                        <extensions>true</extensions>
                        <configuration>
                              <instructions>
                                    <_include>META-INF/MANIFEST.MF</_include>
                                    <Service-Component>OSGI-INF/*</Service-
Component>
                              </instructions>
                        </configuration>
                  </plugin>
            </plugins>
      </build>
```

You also need to apply the following changes in order to migrate the bundles to Karaf:

- From imported packages of the bundles remove the component annotations package as it is only available at compile time: remove org.osgi.service.component.annotations;version="1.3.0"

- Remove the ServiceReference generic arg in the service tracker as Karaf (Felix) implementation does does not provide it

- Remove Service-Component elements from the manifest files of the bundles as it is handled by BND

Navigate to the directories of the three bundles (starting with the Docgen API one) and build them using the following command:

```
mvn clean install
```

You should be able now to install the three bundles in Karaf using the **install** command:

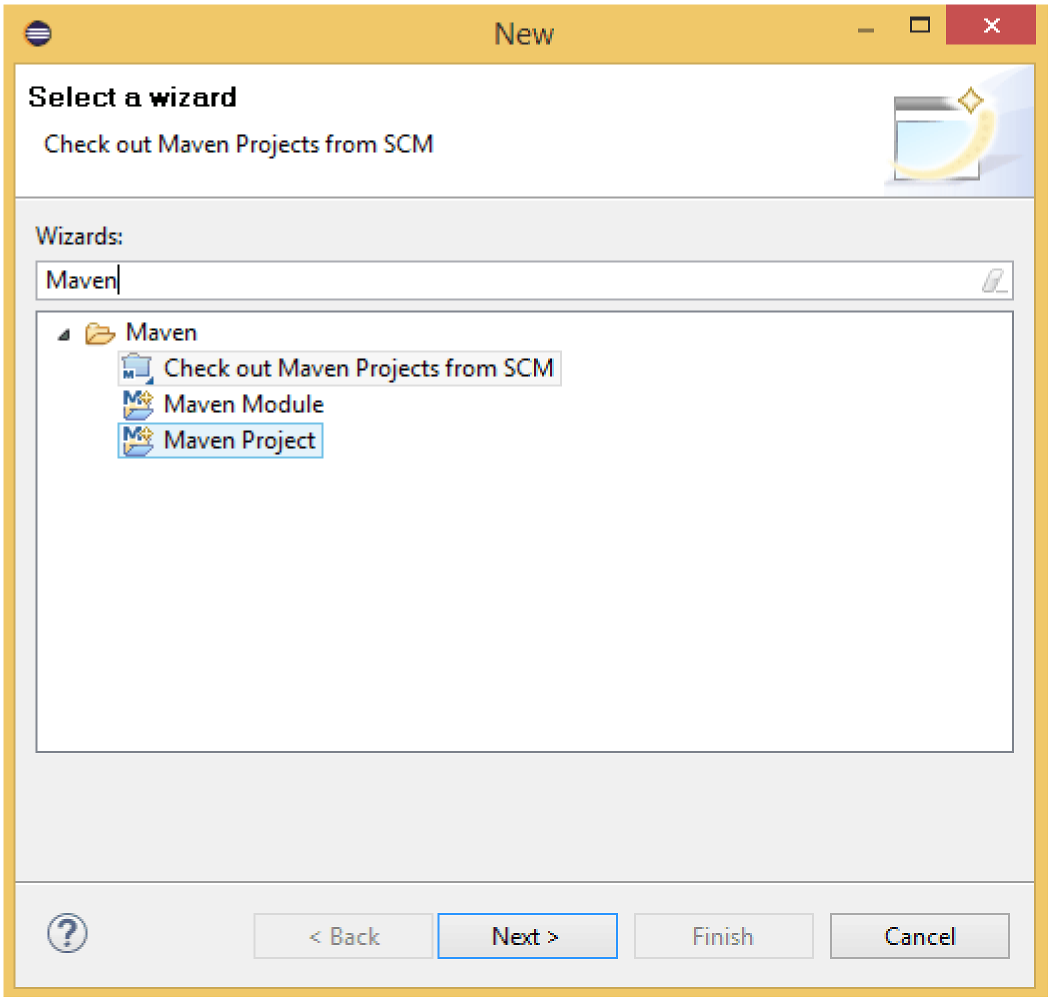
```
install file:///D:/stuff/seminars/OSGi_workshop/workspace/org.osgi.workshop.docgen /target/docgen-1.0.0-SNAPSHOT.jar

install
file:///D:/stuff/seminars/OSGi_workshop/workspace/org.osgi.workshop.docgen.pdf/target/docgen.pdf-1.0.0-SNAPSHOT.jar
```

```
install
file:///D:/stuff/seminars/OSGi_workshop/workspace/org.osgi.workshop.docgen.demo/target/docgen.de
mo-1.0.0-SNAPSHOT.jar
```

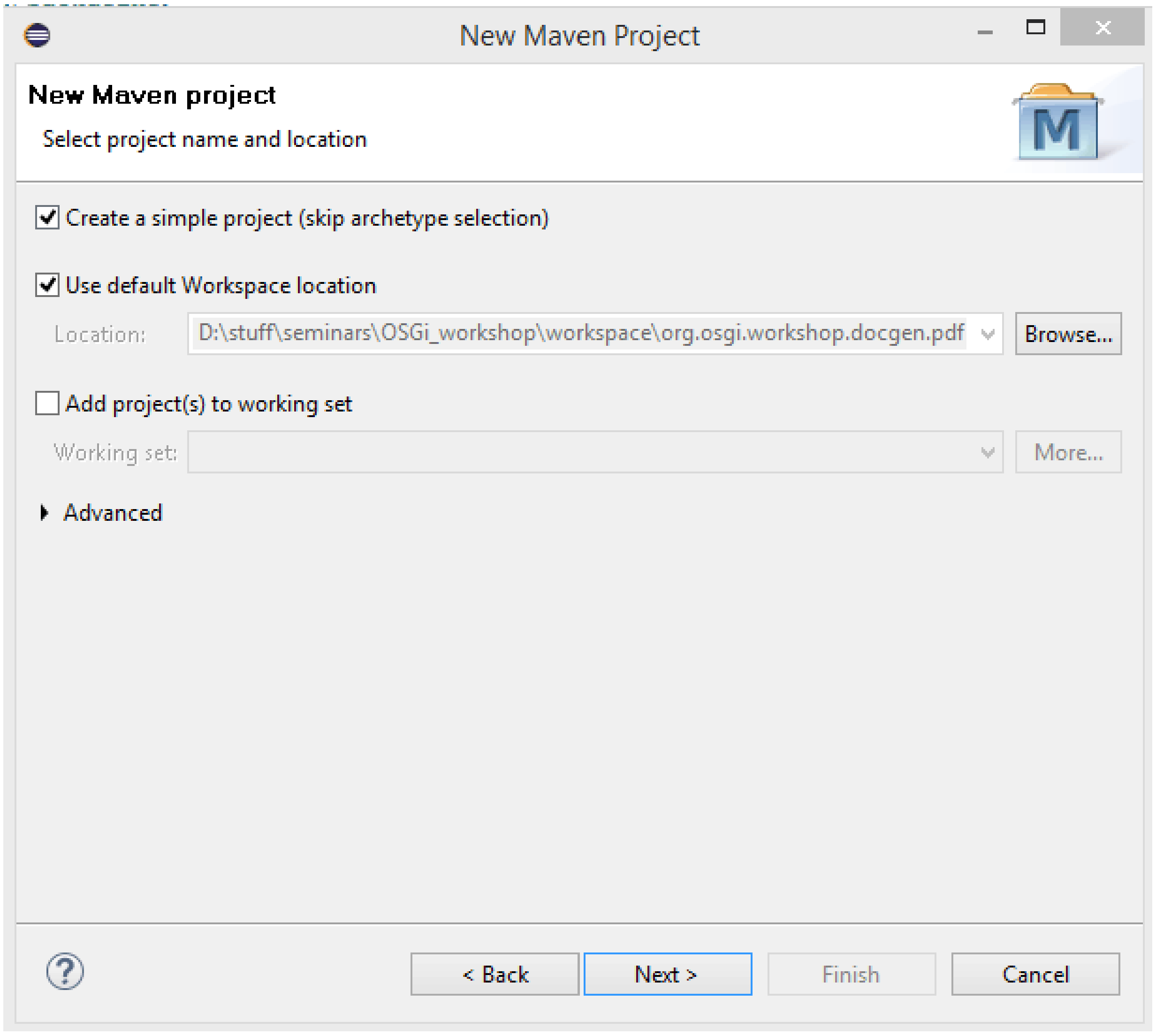
Karaf Maven plug-in

Now we are going to create a Karaf feature build for our bundles and also create a Karaf archive (KAR) that bundles our feature as a Karaf application. Create two Maven projects named **org.osgi.workshop.docgen.feature** and **org.osgi.workshop.docgen.kar**, use **org.osgi.workshop** as the groupID, **docgen.feature** and **docgen.kar** for the artifacIds, **feature** and **kar** for the packaging types.

Select **File** -> **New** -> **Other** and select **Maven Project**:



Click on **Create a simple project (Skip archetype selection):**


New Maven Project

New Maven project

Select project name and location

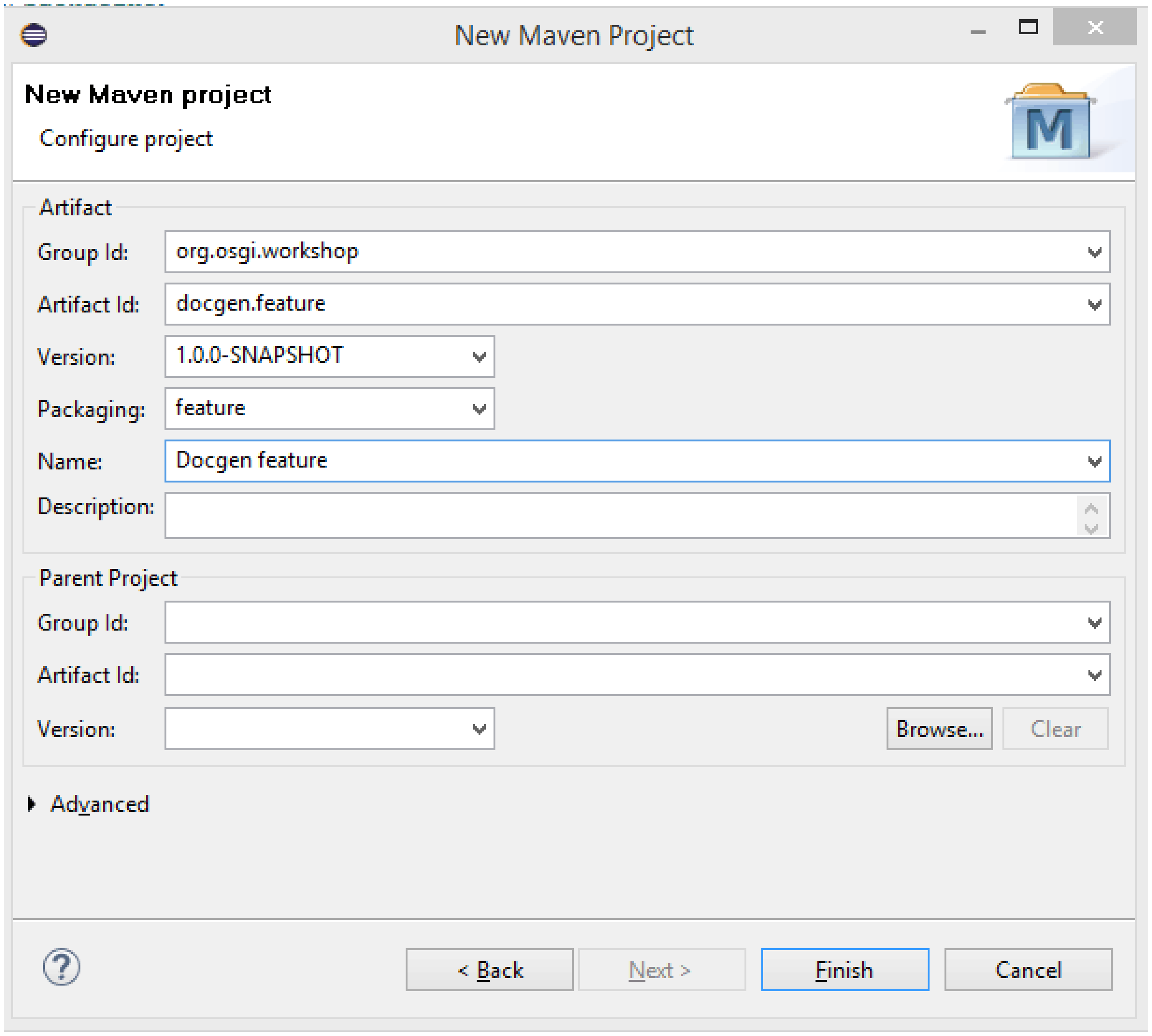
- [✓] Create a simple project (skip archetype selection)
- [✓] Use default Workspace location

Location: D:\stuff\seminars\OSGi_workshop\workspace\org.osgi.workshop.docgen.pdf

- [ ] Add project(s) to working set

Working set:

Provide the following configuration:


New Maven Project

New Maven project

Configure project

Artifact

Group Id: org.osgi.workshop

Artifact Id: docgen.feature

Version: 1.0.0-SNAPSHOT

Packaging: feature

Name: Docgen feature

Description:

Parent Project

Group Id:

Artifact Id:

Version:

Provide the following POM configuration:

```
	<dependencies>
		<dependency>
			<groupId>org.osgi.workshop</groupId>
			<artifactId>docgen</artifactId>
			<version>1.0.0-SNAPSHOT</version>
		</dependency>
		<dependency>
			<groupId>org.osgi.workshop</groupId>
			<artifactId>docgen.pdf</artifactId>
			<version>1.0.0-SNAPSHOT</version>
		</dependency>
		<dependency>
			<groupId>org.osgi.workshop</groupId>
			<artifactId>docgen.demo</artifactId>
			<version>1.0.0-SNAPSHOT</version>
		</dependency>
	</dependencies>
```

```
      <build>
            <plugins>
                  <plugin>
                        <groupId>org.apache.karaf.tooling</groupId>
                        <artifactId>karaf-maven-plugin</artifactId>
                        <version>4.2.1</version>
                        <extensions>true</extensions>
                        <configuration>
                              <enableGeneration>true</enableGeneration>
                        </configuration>
                        <executions>
                              <execution>
                                    <id>generate-features-file</id>
                                    <phase>generate-resources</phase>
                                    <goals>
                                          <goal>features-generate-
descriptor</goal>
                                    </goals>
                              </execution>
                        </executions>
                  </plugin>
            </plugins>
      </build>
```

Then navigate to the directory of the project and execute the following command:

```
mvn clean install
```

You will notice a feature.xml descriptor being generated under the feature/target directory with content similar to the following:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<features xmlns="http://karaf.apache.org/xmlns/features/v1.5.0"
name="docgen.feature">
    <feature name="docgen.feature" description="Docgen feature"
version="1.0.0.SNAPSHOT">
        <feature prerequisite="true" dependency="false">wrap</feature>
        <bundle>mvn:org.osgi.workshop/docgen/1.0.0-SNAPSHOT</bundle>
        <bundle>mvn:org.apache.felix/org.osgi.core/1.0.0</bundle>
        <bundle>mvn:org.osgi.workshop/docgen.pdf/1.0.0-SNAPSHOT</bundle>
        <bundle>mvn:org.osgi.workshop/docgen.demo/1.0.0-SNAPSHOT</bundle>
        <bundle>mvn:org.apache.felix/org.apache.felix.framework/6.0.0</bundle>
        <bundle>wrap:mvn:org.codehaus.mojo/animal-sniffer-
annotations/1.9</bundle>
    </feature>
</features>
```

Now create a Maven project to generate the Karaf application in a KAR archive and provide the following POM configuration:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
      <modelVersion>4.0.0</modelVersion>
      <groupId>org.osgi.workshop</groupId>
      <artifactId>docgen.kar</artifactId>
      <version>1.0.0-SNAPSHOT</version>
      <packaging>kar</packaging>
      <name>Docgen KAR</name>

      <build>
            <plugins>
                  <plugin>
                        <groupId>org.apache.karaf.tooling</groupId>
                        <artifactId>karaf-maven-plugin</artifactId>
                        <version>4.0.0</version>
                        <extensions>true</extensions>
                  </plugin>
            </plugins>
      </build>
</project>
```

Also copy the generated feature.xml file from the docgen.feature project to the src/main/resources/repository directory of the project so that the feature is included in the KAR.

Then run the following:

```
mvn clean install
```

You will notice that a kar file is generated in the target directory. To deploy it run the following in karaf:

```
kar:install file:///D:/stuff/seminars/OSGi_workshop/workspace/docgen.kar/target/docgen.kar-1.0.0-SNAPSHOT.kar
```

Now if you list the installed bundles you will notice that the bundles from the KAR are installed. You can uninstall the KAR (and its bundles) using:

```
kar:uninstall docgen.kar-1.0.0-SNAPSHOT
```

Deploying OSGi bundles

Once we have a set of bundles, either grouped as a feature or as a specific application (such as a Karaf archive) depending on the container we still want to automate the process of deploying those bundles to a target OSGi environment.

Copy the docgen.kar-1.0.0-SNAPSHOT.kar file built in the previous section to the **deploy** directory from the Karaf installation and after a short period of time observe it gets deployed automatically.

Bundle security

We are now going to start Karaf with the default framework security manager and try to deploy and start the docgen and docgen bundles. We will observe that no permissions are granted for imported and exported packages as well as for registration of the PDF service, then we will supply local permissions and re-deploy and activate the bundles.

Stop Karaf and edit the security configuration. Karaf already comes with an etc/all.policy security policy file and possibility to enable framework security manager through the Karaf configuration. Open the etc/system.properties file from the Karaf installation and uncomment the following three properties:

```
java.security.policy=${karaf.etc}/all.policy
org.osgi.framework.security=osgi
org.osgi.framework.trust.repositories=${karaf.etc}/trustStore.ks
```

Now start again Karaf and install the framework-security feature:

```
feature:install framework-security
```

Create the permissions.perm file under the permissions/OSGI-INF folder of the docgen.pdf bundle with the following contents:

```
(org.osgi.framework.PackagePermission "org.osgi.framework" "import")
```

And also include the following under the <build> section of the docgen.pdf bundle:

```
		<resources>
			<resource>
				<directory>permissions</directory>
			</resource>
		</resources>
```

Restart Karaf and try to install the docgen and docgen.pdf bundles:

```
install file:///D:/stuff/seminars/OSGi_workshop/workspace/org.osgi.workshop.docgen /target/docgen-1.0.0-SNAPSHOT.jar

install
file:///D:/stuff/seminars/OSGi_workshop/workspace/org.osgi.workshop.docgen.pdf/target/docgen.pdf-1.0.0-SNAPSHOT.jar
```

You will notice that the docgen.pdf bundle will fail to resolve the docgen.api package. Add it to the permissions.perm file which now looks as follows:

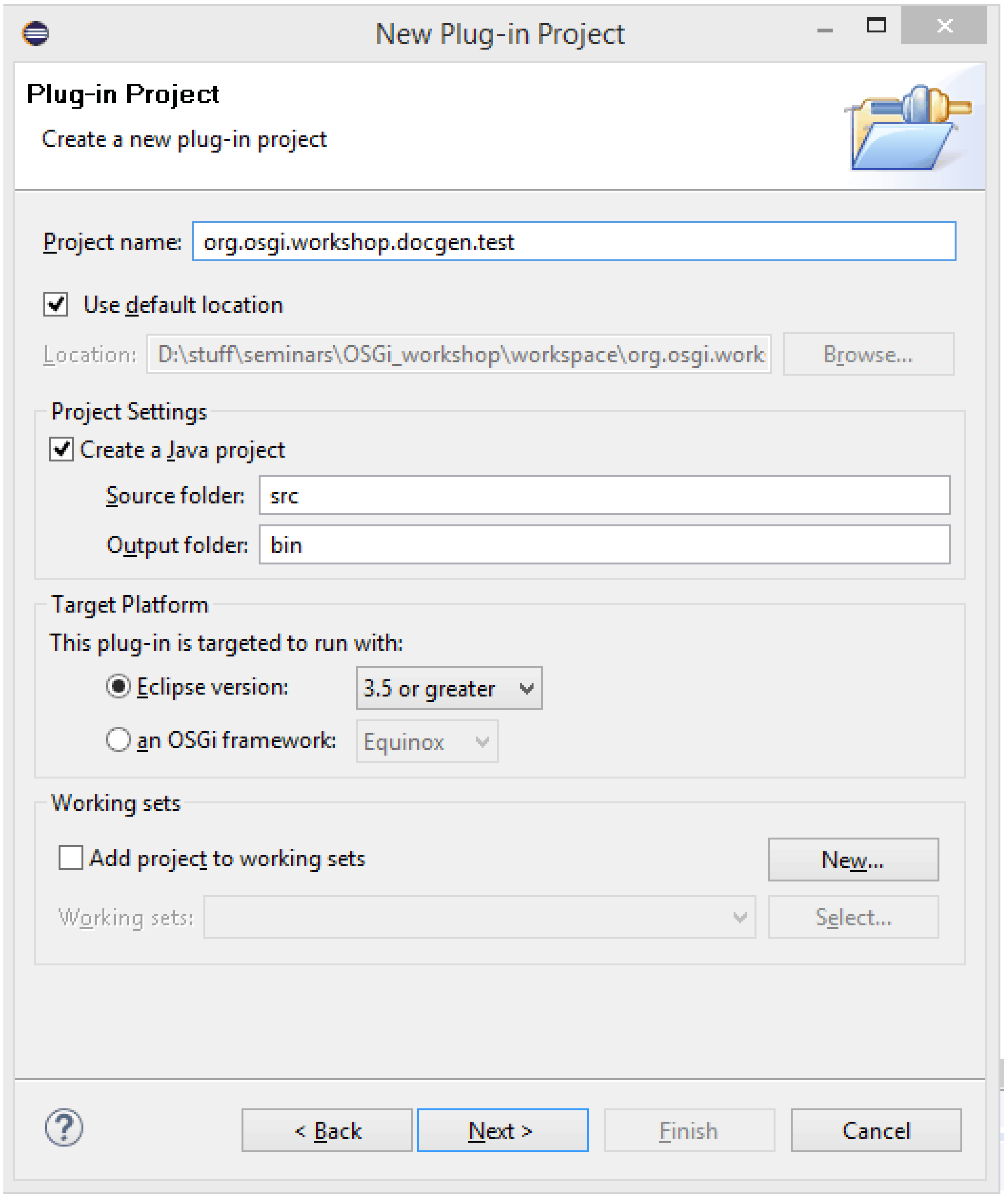
```
(org.osgi.framework.PackagePermission "org.osgi.framework" "import")

(org.osgi.framework.PackagePermission "org.osgi.workshop.docgen.api" "import")
```

Try to reinstall the PDF bundle and you will notice a different error. In the same manner add necessary permissions (or AllPermissions) unti (or AllPermissions) until you are able to resolve the bundle.

Bundle testing

We are going to create integration tests for our bundles using the PaxExam framework. It is a good practice that tests are separated in their own bundles.

Create a new plug-in project named **org.osgi.workshop.docgen.test**:


New Plug-in Project

Plug-in Project

Create a new plug-in project

Project name: org.osgi.workshop.docgen.test

[✓] Use default location

Location: D:\stuff\seminars\OSGi_workshop\workspace\org.osgi.work

Project Settings

[✓] Create a Java project

Source folder: src

Output folder: bin

Target Platform

This plug-in is targeted to run with:

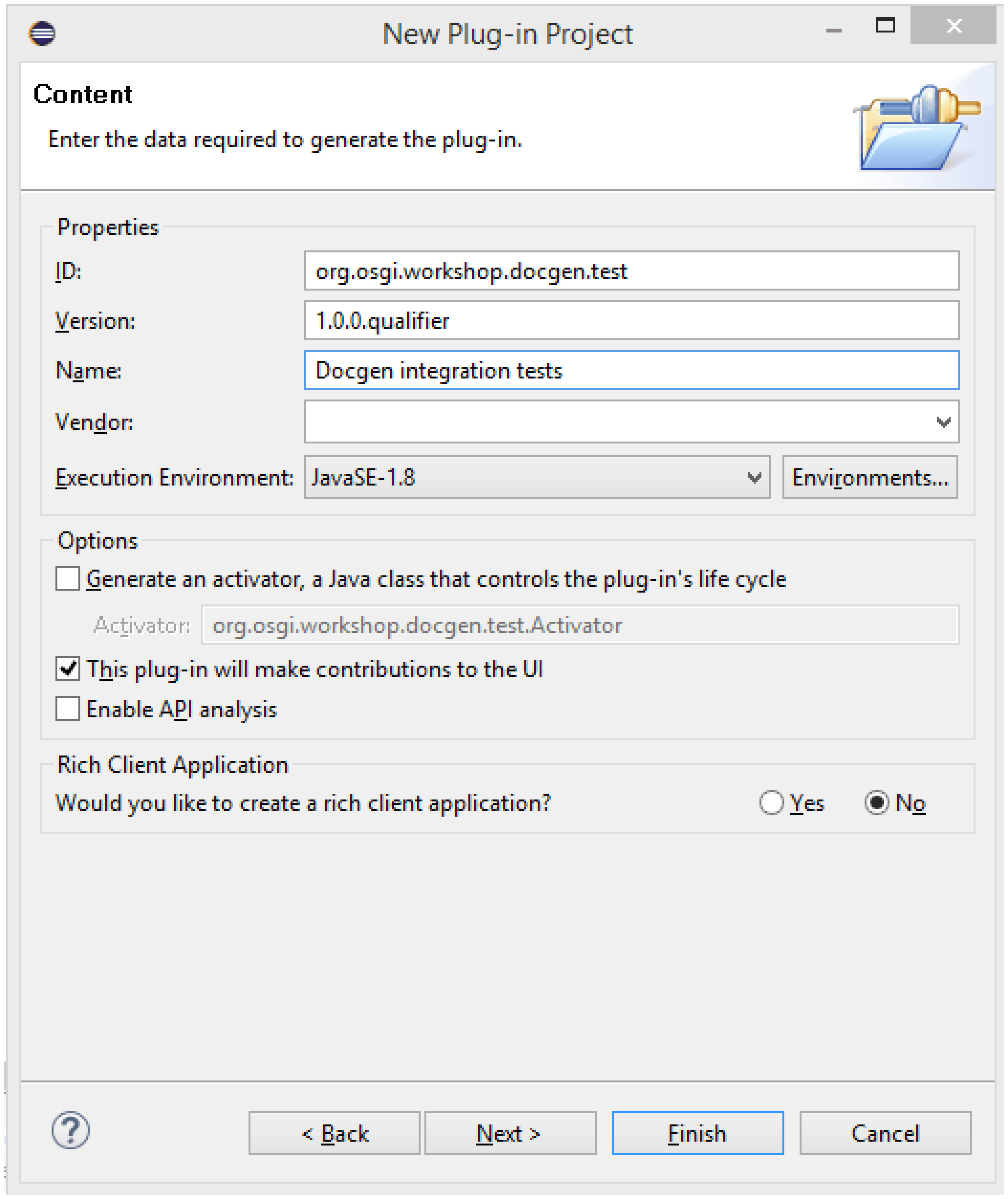
● Eclipse version: 3.5 or greater

○ an OSGi framework: Equinox

Working sets

☐ Add project to working sets

Working sets:



New Plug-in Project

Content

Enter the data required to generate the plug-in.

Properties

ID: org.osgi.workshop.docgen.test

Version: 1.0.0.qualifier

Name: Docgen integration tests

Vendor:

Execution Environment: JavaSE-1.8

Options

- [ ] Generate an activator, a Java class that controls the plug-in's life cycle

Activator: org.osgi.workshop.docgen.test.Activator

- [x] This plug-in will make contributions to the UI
- [ ] Enable API analysis

Rich Client Application

Would you like to create a rich client application? Yes [✓ No]

Then convert it to a Maven project by right-clicking on the project and specifying **Configure** -> **Convert to Maven** project (leave the packaging type empty, **groupId** is org,osgi.workshop and **artifactId** is **docgen.test**).

From the docgen.pdf bundle remove the permissions.perm file and reinstall it in the local Maven repository as follows:

```
mvn clean install
```

Provide the following Maven dependencies for the test bundle:

```
	<dependencies>
```

```
            <dependency>
                  <groupId>org.osgi.workshop</groupId>
                  <artifactId>docgen</artifactId>
                  <version>1.0.0-SNAPSHOT</version>
                  <scope>test</scope>
            </dependency>

            <dependency>
                  <groupId>org.apache.karaf.features</groupId>
                  <artifactId>standard</artifactId>
                  <version>4.2.1</version>
                  <classifier>features</classifier>
                  <type>xml</type>
                  <scope>test</scope>
            </dependency>

            <dependency>
                  <groupId>org.slf4j</groupId>
                  <artifactId>slf4j-api</artifactId>
                  <version>1.7.25</version>
                  <scope>test</scope>
            </dependency>

            <dependency>
                  <groupId>org.slf4j</groupId>
                  <artifactId>slf4j-jdk14</artifactId>
                  <version>1.7.25</version>
                  <scope>test</scope>
            </dependency>

            <dependency>
                  <groupId>org.osgi</groupId>
                  <artifactId>org.osgi.core</artifactId>
                  <version>4.3.0</version>
                  <scope>provided</scope>
            </dependency>

            <!-- Dependencies for pax exam karaf container -->
            <dependency>
                  <groupId>org.ops4j.pax.exam</groupId>
                  <artifactId>pax-exam-container-karaf</artifactId>
                  <version>4.13.1</version>
                  <scope>test</scope>
            </dependency>

            <dependency>
                  <groupId>org.ops4j.pax.exam</groupId>
                  <artifactId>pax-exam-spi</artifactId>
                  <version>4.13.1</version>
                  <scope>test</scope>
            </dependency>

            <dependency>
                  <groupId>org.ops4j.pax.exam</groupId>
                  <artifactId>pax-exam-junit4</artifactId>
                  <version>4.13.1</version>
```

```
                <scope>test</scope>
        </dependency>

        <dependency>
                <groupId>org.ops4j.pax.exam</groupId>
                <artifactId>pax-exam</artifactId>
                <version>4.13.1</version>
                <scope>test</scope>
        </dependency>

        <dependency>
                <groupId>org.ops4j.pax.url</groupId>
                <artifactId>pax-url-aether</artifactId>
                <version>2.5.4</version>
                <scope>test</scope>
        </dependency>

        <dependency>
                <groupId>javax.inject</groupId>
                <artifactId>javax.inject</artifactId>
                <version>1</version>
                <scope>test</scope>
        </dependency>

        <dependency>
                <groupId>junit</groupId>
                <artifactId>junit</artifactId>
                <version>4.12</version>
                <scope>test</scope>
        </dependency>

    </dependencies>
```

And create the PdfDocumentGeneratorTest class under the org.osgi.workshop.docgen.pdf.services package with the following contents:

```
package org.osgi.workshop.docgen.pdf.services;

import static org.ops4j.pax.exam.CoreOptions.maven;
import static org.ops4j.pax.exam.CoreOptions.mavenBundle;

import java.io.File;

import javax.inject.Inject;

import static
org.ops4j.pax.exam.karaf.options.KarafDistributionOption.configureConsole;
import static
org.ops4j.pax.exam.karaf.options.KarafDistributionOption.features;
import static
org.ops4j.pax.exam.karaf.options.KarafDistributionOption.karafDistributionConf
iguration;
import static
org.ops4j.pax.exam.karaf.options.KarafDistributionOption.keepRuntimeFolder;
import org.junit.Assert;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.ops4j.pax.exam.Configuration;
import org.ops4j.pax.exam.ConfigurationManager;
import org.ops4j.pax.exam.Option;
import org.ops4j.pax.exam.junit.PaxExam;
import org.ops4j.pax.exam.options.MavenArtifactUrlReference;
import org.ops4j.pax.exam.options.MavenUrlReference;
import org.osgi.workshop.docgen.api.DocumentGenerator;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

@RunWith(PaxExam.class)
public class PdfDocumentGeneratorTest {

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

      @Inject
      protected DocumentGenerator docgen;

      @Configuration
      public Option[] config() {

            MavenArtifactUrlReference karafUrl =
maven().groupId("org.apache.karaf").artifactId("apache-karaf")
                        .version(karafVersion()).type("zip");

            MavenUrlReference karafStandardRepo =
maven().groupId("org.apache.karaf.features").artifactId("standard")

      .version(karafVersion()).classifier("features").type("xml");

            return new Option[] {
                         karafDistributionConfiguration()
                       .frameworkUrl(karafUrl)
                       .unpackDirectory(new File("target", "exam"))
                       .useDeployFolder(false),
                  keepRuntimeFolder(),
                  configureConsole().ignoreLocalConsole(),
                  features(karafStandardRepo , "scr"),

      mavenBundle().groupId("org.osgi.workshop").artifactId("docgen").version(
"1.0.0-SNAPSHOT").start(),

      mavenBundle().groupId("org.osgi.workshop").artifactId("docgen.pdf").vers
ion("1.0.0-SNAPSHOT").start(),

            };
      }

      public static String karafVersion() {
            ConfigurationManager cm = new ConfigurationManager();
            String karafVersion = cm.getProperty("pax.exam.karaf.version",
"4.2.1");
            return karafVersion;
      }
```

```
	@Test
	public void testAdd() {
		docgen.generate("D:\\test11.pdf", "Pax Exam test");
		Assert.assertTrue(true);
	}

}
```

The Karaf configuration dictates that Pax Exam downloads a Karaf distribution under a folder that you specify and provisions additional configuration and bundles before running the test. By default this is done on every test but we can specify a different reactor strategy (with the @ExamReactoryStrategy annotation on the test class). When the downloaded Karaf runtime starts and bundles are provisioned the Pax Exam probe bundle that contains the current tests are executed within the Karaf runtime. The test class injects an instance of the org.osgi.workshop.docgen.api.DocumentGenerator service and uses it to generate a sample PDF file.

Run the test directly from Eclipse and make sure that the test11.pdf file is generated with the specified content.

Debugging

Diagnostic commands

In the Karaf shell run the following command to create a runtime dump:

```
dev:dump-create
```

Observe the contents of the created zip file under the Karaf installation.

Run the following command to determine why certain bundles cannot be resolved:

```
bundle:diag
```

Display the dependency tree for the docgen and docgen.pdf bundles:

```
bundle:tree-show  org.osgi.workshop.docgen
bundle:tree-show org.osgi.workshop.docgen.pdf
```

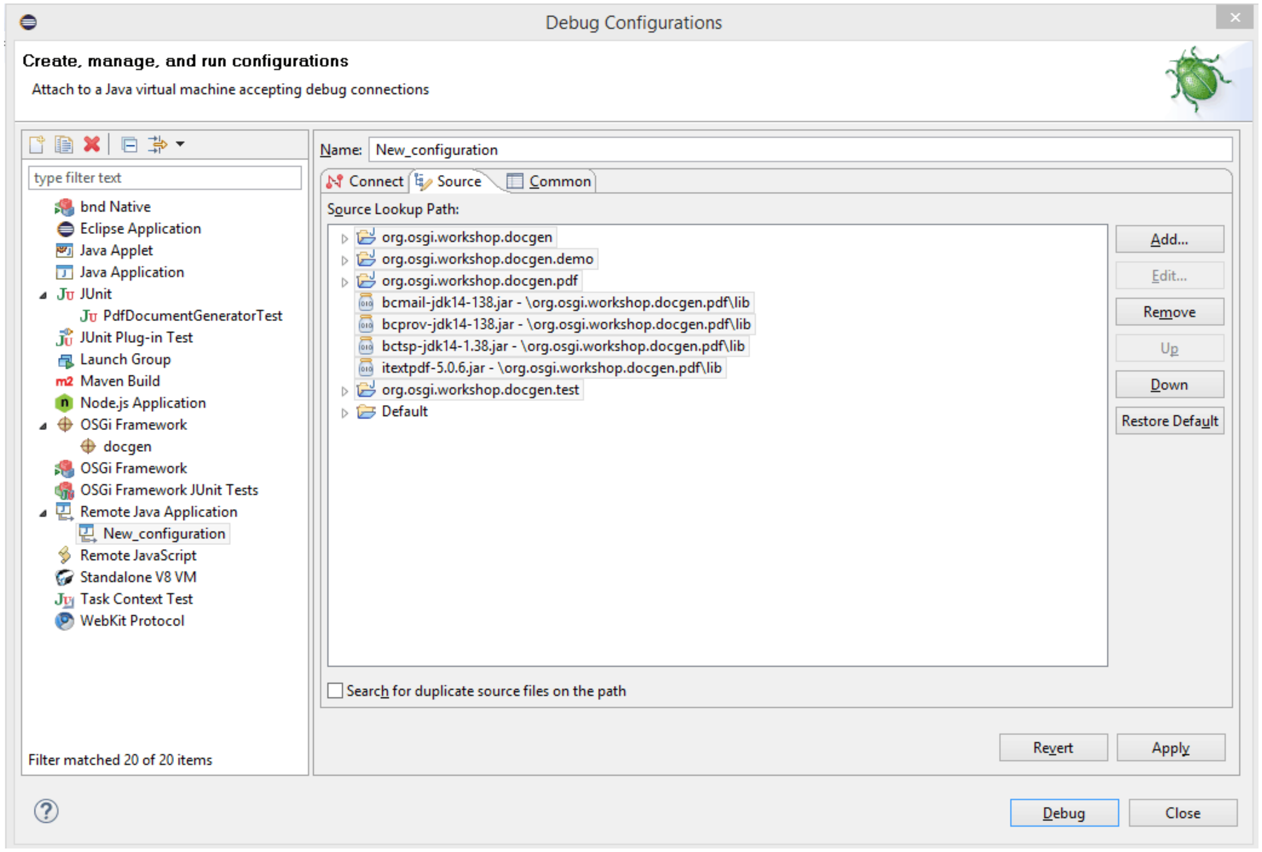
Remote debugging

Restart the Karaf runtime with the debug option to enable remote debugging:

```
bin\karaf.bat debug
```

The remote debug port used by default from Karaf is 5005.

Next create a remote debug session from Eclipse by creating a new debug configuration on port 5005 and adding the projects as sources:


Debug Configurations

Create, manage, and run configurations

Attach to a Java virtual machine accepting debug connections

type filter text

- bnd Native
- Eclipse Application
- Java Applet
- Java Application
- JUnit
  - PdfDocumentGeneratorTest
- JUnit Plug-in Test
- Launch Group
- Maven Build
- Node.js Application
- OSGi Framework
  - docgen
- OSGi Framework
- OSGi Framework JUnit Tests
- Remote Java Application
  - New_configuration
- Remote JavaScript
- Standalone V8 VM
- Task Context Test
- WebKit Protocol

Filter matched 20 of 20 items

Name: New_configuration

Connect | Source | Common

Source Lookup Path:

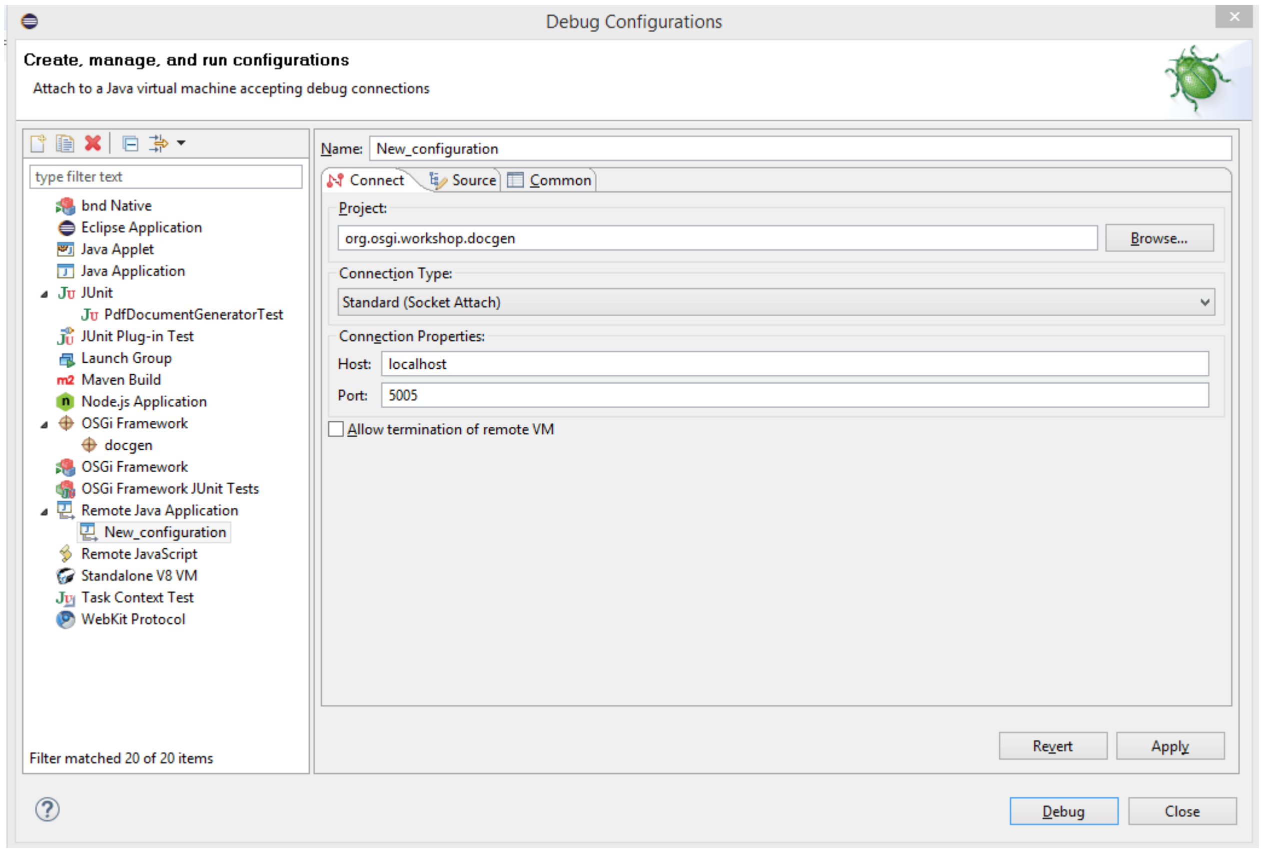
- org.osgi.workshop.docgen
- org.osgi.workshop.docgen.demo
- org.osgi.workshop.docgen.pdf
- bcmail-jdk14-138.jar - \org.osgi.workshop.docgen.pdf\lib
- bcprov-jdk14-138.jar - \org.osgi.workshop.docgen.pdf\lib
- bctsp-jdk14-1.38.jar - \org.osgi.workshop.docgen.pdf\lib
- itextpdf-5.0.6.jar - \org.osgi.workshop.docgen.pdf\lib
- org.osgi.workshop.docgen.test
- Default

Add... | Edit... | Remove | Up | Down | Restore Default

☐ Search for duplicate source files on the path

Revert | Apply

Debug | Close


Debug Configurations

Create, manage, and run configurations

Attach to a Java virtual machine accepting debug connections

Name: New_configuration

Project: org.osgi.workshop.docgen

Connection Type: Standard (Socket Attach)

Connection Properties:

Host: localhost

Port: 5005

Allow termination of remote VM

Filter matched 20 of 20 items

After running the debug configuration, put a breakpoint in the start method of the docgen activator, restart the bundle and observe that Eclipse stops on the breakpoint.

You can also dynamically update the bundles by running the bundle:watch command and every time you rebuild the bundle from the same location it gets updated in Karaf:

```
bundle:watch *
```

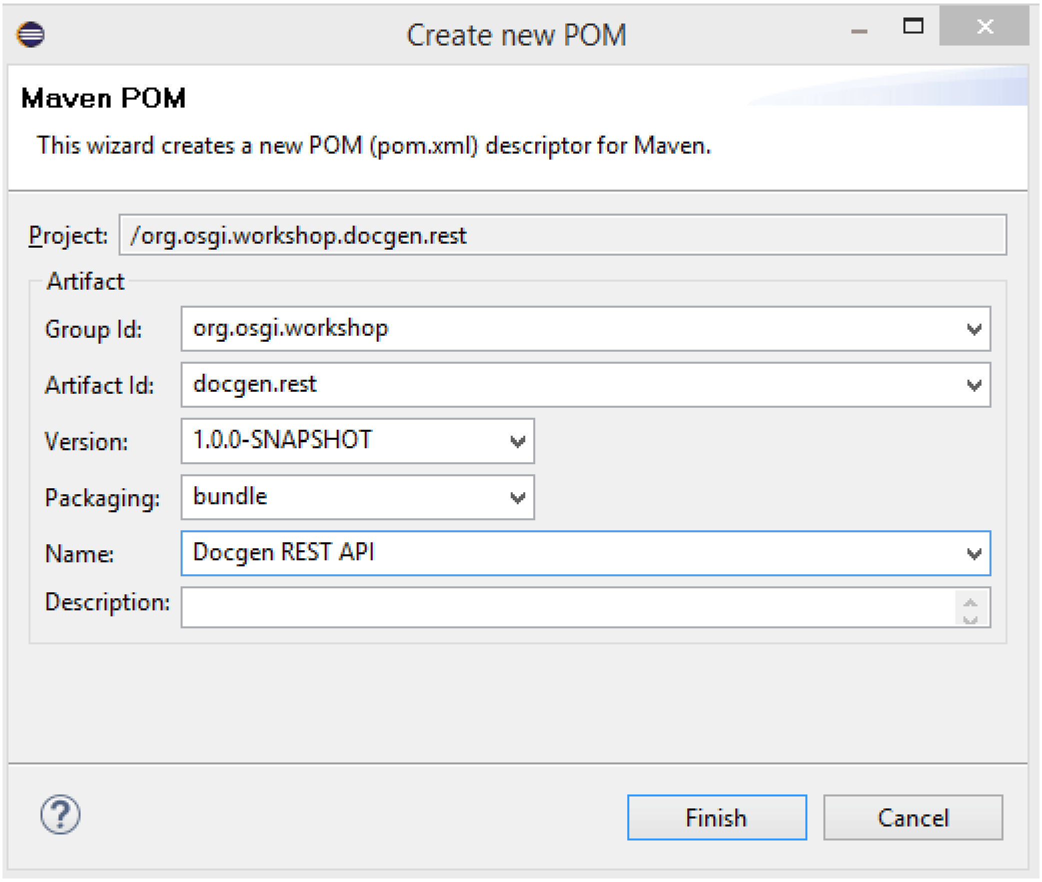
OSGi compendium and additional services

REST Service

We are now going to expose a REST service that allows us to generate PDF documents from a REST endpoint. To do so we will use the bundles from the OSGi – RAX-RS Connector project (https://github.com/hstaudacher/osgi-jax-rs-connector). Install the four bundles (jersey-min, publisher, consumer and servlet-api provided from the jax-rs-connector feature or use the bundle pack coming with the tutorial).

```
Install file:///consumer-5.3.1.jar
Install file:///jersey-min-2.22.2.jar

Install file:///publisher-5.3.1.jar

Install file:///javax.servlet-api-3.0.1.jar
```

Next create a project called org.osgi.workshop.docgen.rest as an Eclipse plug-in project and convert it to a Maven project:



Maven POM

This wizard creates a new POM (pom.xml) descriptor for Maven.

Project: /org.osgi.workshop.docgen.rest

Artifact

Group Id: org.osgi.workshop

Artifact Id: docgen.rest

Version: 1.0.0-SNAPSHOT

Packaging: bundle

Name: Docgen REST API

Description:

Add the following configuration for the Maven project (dependencies and maven-bundle-plugin configuration):

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
      <modelVersion>4.0.0</modelVersion>
      <groupId>org.osgi.workshop</groupId>
      <artifactId>docgen.rest</artifactId>
      <version>1.0.0-SNAPSHOT</version>
      <packaging>bundle</packaging>
      <name>Docgen REST API</name>

      <dependencies>
```

```
			<dependency>
				<groupId>org.osgi.workshop</groupId>
				<artifactId>docgen</artifactId>
				<version>1.0.0-SNAPSHOT</version>
			</dependency>

			<dependency>
				<groupId>javax.ws.rs</groupId>
				<artifactId>javax.ws.rs-api</artifactId>
				<version>2.0.1</version>
			</dependency>

			<dependency>
				<groupId>com.eclipsesource.jaxrs</groupId>
				<artifactId>jersey-min</artifactId>
				<version>2.22.2</version>
			</dependency>
			<dependency>
				<groupId>com.eclipsesource.jaxrs</groupId>
				<artifactId>publisher</artifactId>
				<version>5.3.1</version>
			</dependency>
			<dependency>
				<groupId>com.eclipsesource.jaxrs</groupId>
				<artifactId>consumer</artifactId>
				<version>5.3.1</version>
			</dependency>
			<dependency>
				<groupId>javax.servlet</groupId>
				<artifactId>servlet-api</artifactId>
				<version>3.0-alpha-1</version>
			</dependency>
			<dependency>
				<groupId>org.osgi</groupId>

		<artifactId>org.osgi.service.component.annotations</artifactId>
				<version>1.3.0</version>
			</dependency>
		</dependencies>

		<build>
			<sourceDirectory>src</sourceDirectory>
			<plugins>
				<plugin>
					<groupId>org.apache.felix</groupId>
					<artifactId>maven-bundle-plugin</artifactId>
					<extensions>true</extensions>
					<configuration>
						<instructions>
							<Bundle-SymbolicName> ${pom.artifactId}
							</Bundle-SymbolicName>
							<Bundle-Name>${pom.artifactId}</Bundle-
Name>
							<Bundle-Version>1.0.0</Bundle-Version>
							<Bundle-
Activator>org.osgi.workshop.docgen.rest.Activator</Bundle-Activator>
							<Import-Package>
```

```
                                                javax.ws.rs.*,
                                                org.osgi.util.tracker,
                                                org.osgi.service.log,
                                                org.osgi.framework,
                                                org.osgi.workshop.docgen.api,
                                          </Import-Package>
                                          <_include>META-INF/MANIFEST.MF</_include>
                                          <Service-Component>OSGI-INF/*</Service-
Component>
                                          <Bundle-ClassPath>.</Bundle-ClassPath>
                                    </instructions>
                              </configuration>
                        </plugin>
                  </plugins>
      </build>
</project>
```

Under the org.osgi.workshop.docgen.rest.api package create the DocgenServiceEndpoint endpoint class with the following definition:

```
package org.osgi.workshop.docgen.rest.api;

import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.QueryParam;
import javax.ws.rs.core.Response;

@Path("api")
public interface DocgenServiceEndpoint {

      @POST
      @Path("pdf")
      @Produces("application/pdf")
      public Response generatePdf(@QueryParam("filename") String filename,
@QueryParam("test") String text);
}
```

Under the same package add the implementation class DocgenServiceEndpointImpl as follows:

```
package org.osgi.workshop.docgen.rest.api;

import java.io.File;

import javax.ws.rs.core.Response;
import javax.ws.rs.core.Response.ResponseBuilder;

import org.osgi.service.component.annotations.Component;
import org.osgi.service.component.annotations.Reference;
import org.osgi.service.component.annotations.ReferenceCardinality;
import org.osgi.service.component.annotations.ReferencePolicy;
import org.osgi.workshop.docgen.api.DocumentGenerator;
```

```
@Component
public class DocgenServiceEndpointImpl implements DocgenServiceEndpoint {

	private DocumentGenerator docgen;

	@Reference(name = "docgen", service = DocumentGenerator.class,
cardinality = ReferenceCardinality.MANDATORY, policy = ReferencePolicy.STATIC,
unbind = "unbindDocumentGenerator")
	public void bindDocumentGenerator(DocumentGenerator docgen) {
		this.docgen = docgen;
	}

	public void unbindDocumentGenerator(DocumentGenerator docgen) {
		this.docgen = null;
	}


	public Response generatePdf(String filename, String text) {
		docgen.generate(filename, text);
		File file = new File(filename);
	    ResponseBuilder response = Response.ok((Object) file);
	    response.header("Content-Disposition","attachment;
filename=generated.pdf");
	    return response.build();
	}

}
```

Once the DocgenServiceEndpointImpl service implementation is registered to the service registry the JAX-RS connector scans the service for JAX-RS annotations (as we have provided them in the interface) and automatically registers the services under the /services context root.

Next build the bundle, install it and start it into Karaf. To test that it works run the following (CURL must be installed):

```
curl –X POST "localhost:8181/services/api/pdf?filename="D:\\test20.pdf"&test=testREST"
```

Observe that the test2.pdf file is created with the provided text.

References

OSGi Alliance
http://www.osgi.org/Main/HomePage

OSGi Specifications
https://www.osgi.org/developer/specifications/

Modularity - what is it ?
http://www.infoq.com/articles/modular-java-what-is-it/

Java modularity - why ?
http://java.dzone.com/articles/java-modularity-2-why

Java JAR hell problem
http://en.wikipedia.org/wiki/Java_Classloader#JAR_hell

Java 8 Modules Jigsaw and OSGi
http://www.slideshare.net/mfrancis/java-8-modules-jigsaw-and-osgi-neil-bartlett

Java Modularity - OSGi and Project Jigsaw
http://techdistrict.kirkk.com/2009/06/12/java-modularity-osgi-and-project-jigsaw/

OSGi concepts
https://www.eclipse.org/virgo/documentation/virgo-documentation-3.7.0.M01/docs/virgo-user-guide/html/ch02s02.html

OSGi Modularity tutorial (Vogella)
http://www.vogella.com/tutorials/OSGi/article.html

OSGi Alliance: Where to Start
https://www.osgi.org/developer/where-to-start/

OSGi Alliance: Tutorial archive
https://www.osgi.org/learning-resources-tutorials/tutorial-archive/

Introduction to OSGi (Baeldung)
https://www.baeldung.com/osgi

OSGi services tutorial
http://www.javasavvy.com/osgi-services-tutorial/

OSGi introductory tutorial (by Peter Kriens)
http://archive.oredev.org/download/18.5bd7fa0510edb4a8ce4800019273/Peter_Kriens_-_Workshop_OSGi_Hands-on.pdf

OSGi tutorial with Knoplerfish
https://www.cs.ucy.ac.cy/~cs00pe/epl603/labs/osgi_tutorial.pdf

OSGi enRoute
https://enroute.osgi.org/

Developing OSGi Enterprise applications workshop
https://www.osgi.org/wp-content/uploads/OSGiCE-OSGiFreeToolsLab.pdf

OSGi getting started tutorial
https://mnlipp.github.io/osgi-getting-started/

BND Tools tutorial
https://bndtools.org/tutorial.html

Sonatype p2 repositories
https://help.sonatype.com/repomanager2/p2-repositories

Apache Karaf documentation
https://karaf.apache.org/manual/latest/

Karaf remote management in Eclipse
https://wiki.eclipse.org/Karaf_Remote_Management_with_Eclipse

Maven bundle plug-in
http://felix.apache.org/documentation/subprojects/apache-felix-maven-bundle-plugin-bnd.html

BND Maven plug-in
https://github.com/bndtools/bnd/tree/master/maven/bnd-maven-plugin

OSGi security
http://moi.vonos.net/java/osgi-security/

Building Secure OSGi applications
https://cwiki.apache.org/confluence/download/attachments/7956/Building+Secure+OSGi+Applications.
pdf