A Guide to Developing OSGi applications with Osmorc in IntelliJ IDEA

Robert F. Beeger

A Guide to Developing OSGi applications with Osmorc in IntelliJ IDEA

Robert F. Beeger

Table of Contents

List of Figures

List of Examples

Preface

Developing OSGi applications with IDEA

Osmorc is a plugin for IntelliJ IDEA that adds support for OSGi to the IDE. This book guides the reader from first steps like setting up a project in IDEA that uses OSGi through all areas of the OSGi support to more specific tasks like running and debugging Eclipse RCP applications.

This book is not an introduction to OSGi. It's sole purpose is to guide people who know and understand OSGi to the specific support for OSGi in IDEA.

Two approaches to the development of OSGi applications

I started developing Osmorc in autumn 2007 to enable me to develop Eclipse RCP applications with IDEA. The name “Osmorc” is a somewhat deliberate abbreviation of “OSGi Module Layer and Eclipse RCP Support”. That longer variant also describes the first driving force in the development of the plugin. The first goal was to somehow map the dependencies defined in the manifests of the bundles in a project to IDEA module dependencies. And finally I wanted to be able to develop and run Eclipse RCP applications in IDEA. We will deal with those goals in the second part of the book.

In May 2008 I was joined by Jan Thomä who had a completely different approach to the development of OSGi applications. He used Maven and IDEA's module system to take care of the dependencies between bundles and generated the bundle manifests using BND. So he used OSGi only as a runtime environment. His main focus in the development of Osmorc is on generating manifest files from the structure of a project and the integration of Maven. The “generating” aspect of Osmorc and the integration of Maven with it will be described in the third part of this book.

It's important to understand that those are two different approaches to developing OSGi applications. Which of those you choose depends on various parameters. The simplest is preference. People migrating their Eclipse RCP applications will probably prefer the first one. That is also the way preferred by people wanting to have total control over all OSGi aspects of their applications.

People knowing and loving Maven will prefer the second way. They can keep on developing their applications with Maven and will gain the possibility to directly run their applications with OSGi from IDEA.

Other factors may also be important when choosing one of the two approaches. One possible other factor is the used build system. Continuous integration servers like TeamCity and Hudson directly support building Maven projects while building OSGi applications currently requires writing Ant build scripts or something like that.

Structure of the book

The first part of the book deals with concepts and other things common to the two approaches. Among others it shows how to setup a framework instance.

The second part of the book deals with the first of the two mentioned approaches. It begins with the basics like setting up a fresh project for an OSGi application and migrating an existing Eclipse RCP project to IDEA and proceeds on to advanced tasks like running and debugging an OSGi application in IDEA.

The third part shows the second approach in action.

Part I. Commons

This part shows concepts and areas of Osmorc that are common to both approaches to developing OSGi applications with IDEA.

Chapter 1. Defining framework instances

Though it's not a requirement, defining at least one framework instance before anything else OSGi related with Osmorc is a good idea. And so that's where we start.

What is a framework instance?

OSGi itself is a specification. It doesn't provide an implementation. There are several frameworks implementing the OSGi specification. Osmorc currently supports Eclipse Equinox, Apache Felix, Knopflerfish and Concierge OSGi.

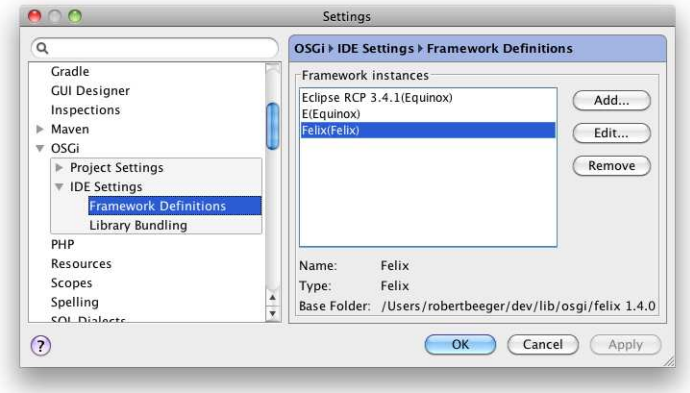
Each of those implementations is able to run a basic OSGi application that uses only the standard features and interfaces defined in the OSGi specifications. Each implementation may add proprietary features that only run on that implementation. Eclipse Equinox for example provides several specific features used in Eclipse RCP applications that are not found in any other implementation.

An implementation is often just called a *framework*. Since you can have several different installations or instances of a framework installed on your computer, we call one specific installation a *framework instance*. You could have one instance of Equinox that is actually the Eclipse RCP SDK and one that is the pure Equinox framework without SWT or any other RCP specific bundles. You could have one instance for version 3.3.0 of Equinox and one for version 3.5.0. It's also possible to have an instance that contains specific bundles needed by your project.

Installing a framework instance

To setup a framework instance you need to open the settings of IDEA (File → Settings). In the “Project Settings” section you'll find a node labeled “OSGi”. Now we are only interested in the subnode “Framework Definitions”, which is shown in Figure 1.1.

Figure 1.1. Framework instance definitions

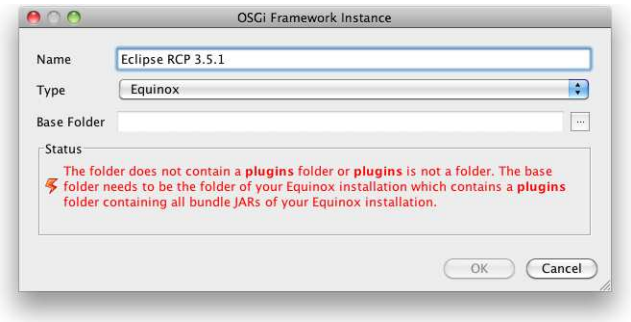


An important detail to note is that framework instances are only defined for a given IDEA installation. They are not part of a project.

The UI here is very simple. The list at the top contains the already defined framework instances and details of the selected instance are shown below. Currently those details are sparse Three buttons to the right of the list are used for adding new instances and editing or removing the currently selected instance. There's nothing special about the “Remove” button and the details you can change with the

"Edit" button are the same that you specify when adding a new instance. So let's see what the "Add" button is about. Clicking on it brings up the dialog shown in Figure 1.2

Figure 1.2. Adding a framework instance



The first thing to specify about a new framework instance is its name. So if you are creating a framework instance definition for the Eclipse RCP SDK 3.5, you would probably call it "Eclipse RCP 3.5". If you have a customized version with for example Spring, you could call it "Eclipse RCP 3.5 with Spring DM".

Osmorc currently supports the Equinox, Felix, Knopflerfish and Concierge implementations of OSGi. Since each of those implementations have different layouts of their installation directories and different ways of starting them, Osmorc needs to know what type of framework instance is being specified. So that's what you need to define next.

The last thing to specify is the base folder. That's the folder where you installed or unpacked the framework to. As you can see in Figure 1.2 an error text is shown if no folder or a wrong folder is chosen, telling you what structure the folder is expected to have. In the following section we'll look into each kind of framework installation. If you want to add bundles to a framework and you also want to add the sources of those added bundles, you need to add them in a way that is recognizable for the specific framework type.

Installing Eclipse Equinox

Equinox is the OSGi implementation of the Eclipse Foundation. It is also the base for the Eclipse RCP SDK. So whether you are just using the basic OSGi functionality or the RCP SDK, use the "Equinox" type.

In the typical layout the installation folder contains a "plugins" folder that contains all bundles and also the matching sources.

The Equinox SDK normally also contains the sources of its bundles. There are currently two formats of sources Osmorc recognizes for Equinox.

1. The old format of Equinox 3.3 and earlier. In this case there is a folder whose name contains the string "source". That folder contains a folder called "src" which then contains folders with the names of the bundles. Each of those folders contains a file src.zip which contains the sources for that bundle

2. In Equinox 3.4.0 source bundles were introduced. So for a bundle "org.eclipse.osgi_3.4.2.R34x_v20080826-1230.jar" there is a source bundle "org.eclipse.osgi.source_3.4.2.R34x_v20080826-1230.jar". Equinox 3.4.0 introduced the "Eclipse-SourceBundle" manifest header to mark those specific bundles. Osmorc currently does not know this header. Osmorc recognizes source bundles by their names. For a bundle org.foo_1.0.0.jar it will try to load the sources from org.foo.source_1.0.0.jar.

Osmorc is able to start Equinox frameworks starting from version 3.1.0 up to the latest currently available version 3.5.0

Installing Apache Felix

The folder of a Felix installation should contain a bin folder and a bundle folder. That's the standard structure of the distribution. The distribution does not contain the source code of the framework. It can be downloaded separately.

Since there is no standard place for source files in the Felix distribution, Osmorc searches for sources in the subfolder src, which should contain a bin and a bundle folder. In those folders Osmorc searches for zip files containing the sources. So for org.osgi.core-1.2.0 in bundle it will try to load src/bundle/org.osgi.core-1.2.0-project.zip. And inside those zips it will load the folder "/src/main/java". That's all a bit complicated but that's the way source zips are named and structured in Felix.

Installing Knopflerfish

Just specify the folder where you installed Knopflerfish into. This folder will contain a folder knopflerfish.org that contains a folder osgi with a folder *jars* containing the bundle jars and a folder *bundles* containing the sources.

Installing Concierge

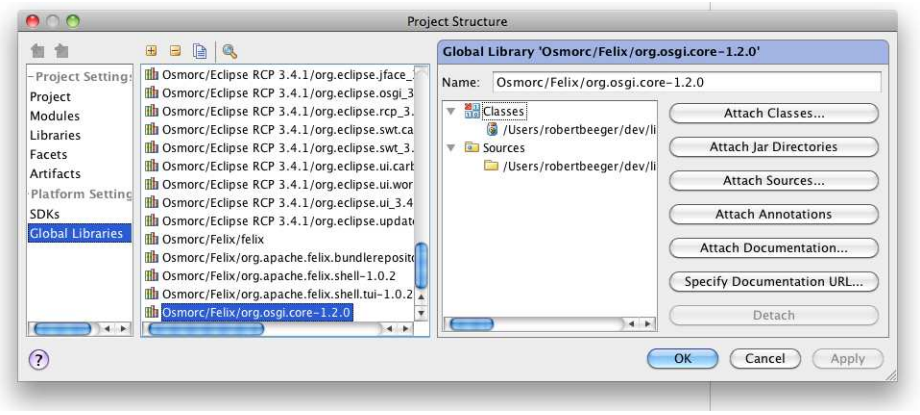
The installation folder of a Concierge framework should contain a folder bundles containing the bundles of the framework. The Concierge distribution doesn't contain sources. Those need to be downloaded separately. Osmorc assumes that sources are contained in zip files having the same names as their matching bundles. Those zip files are searched for in a subfolder named *src*.

Manual tweaking of framework instances

Currently the support for the definition of framework instances is very basic. Better support will be implemented in the future, but for the time being, it may sometimes be necessary to manually tweak the framework instances. If for some reason the sources of a bundle are not found or if it is too much of a hassle to convert them into a format that is recognized by Osmorc, tweaking may help. Also tweaking can be used to add additional bundles that cannot be found in the standard folder structure of the framework.

For each bundle of a framework Osmorc creates a global library (Take a look at File → Project Structur → Global Libraries).

Figure 1.3. The libraries of a framework



As you can see in Figure 1.3 those libraries have names that follow a distinctive pattern, which is "Osmorc/frameworkName/bundleJarName". frameworkName is the name of the framework to which a library belongs. bundleJarName is the name of the jar of the bundle that is being referenced.

You can attach sources to framework libraries for which Osmorc failed to find the sources. You can also add new libraries and so new bundles to a framework by creating new global libraries that follow the shown naming scheme. Whenever you add new bundles in that way, you need to restart IDEA.

Part II. Direct Manifest Manipulation

This part introduces the basic concepts in Osmorc and shows how developing OSGi applications is supported when the manifest files are manually maintained by the developer.

Chapter 2. First steps

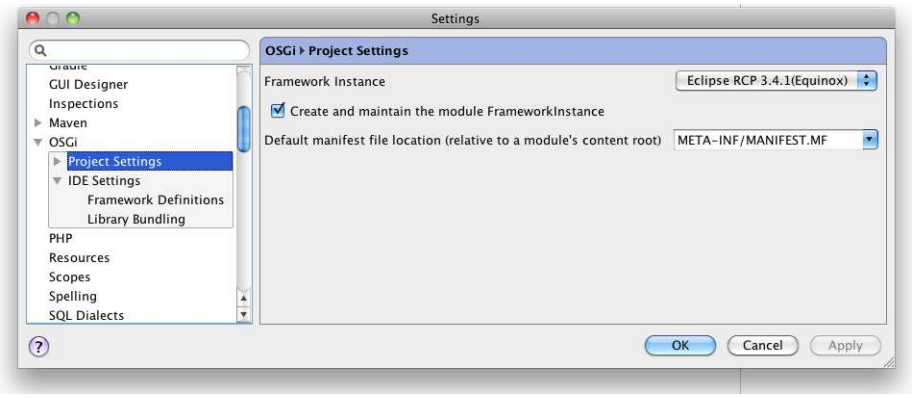
We start with a basic setup. We'll keep it simple. The goal of this chapter is to see a BundleActivator say “Hello world!” on IDEA's run console.

To get us started we create a project named “HelloWorldOSGi” with one java module named “example.helloworld”. OSGi bundles normally get the name of their main package as their symbolic name. It's convenient but not mandatory to name an IDEA module for an OSGi bundle in the same manner. So that's the reason for the name of the module.

Adjusting project settings for an OSGi project

The first thing we'll do is make some adjustment to the project settings. Those are settings that affect the whole project and each module with an OSGi facet. So we open File → Settings → Project Settings → OSGi → Project Settings. Figure 2.1 shows IDEA's Settings dialog at that point.

Figure 2.1. OSGi project settings



As a first we choose a framework instance that is used for the project. The framework instance you choose here is not necessarily the one that you later use to run your OSGi application on. Typically though that will be the case, especially if you use proprietary features of the framework instance in you application. This setting basically makes sure that you can import packages from the framework into your bundles and use the classes contained in them. We have chosen an Equinox instance here, but for this example you can use any framework instance of any kind since we don't use any Equinox-specific features.

As this is a project setting it will be shared if you check the project into a version control system. Once you check it out on another computer where the chosen framework instance is not yet defined, it will be marked as undefined on the project settings page. You can then add it as described in Chapter 1. Osmorc will propose the name of the undefined instance as the name of the framework instance to be added. So you don't need to remember how exactly it is written.

Enabling “Create and maintain the module FrameworkInstance” will cause a new module with the name “FrameworkInstance” to be created. This module references all bundle jars of the framework instance used for the project. Why would you need such a module? As long as you don't import any package from a framework module, the corresponding global library isn't referenced anywhere in your project. As long as a global library isn't referenced in a project, you won't be able to navigate to it by “Go To Class” action. So when your framework instances contain some specific bundles - Spring or SWT for example -, you won't be able to search for classes inside those bundles. That's where that new module comes in. It references all global libraries belonging to your chosen framework and makes

them searchable from the project. You will find them with the “Go To Class” or you can just browse through all available bundles by browsing through the libaries of that special module. It's definitely not the best way to present the contents of a framework instance, but for the time being it works.

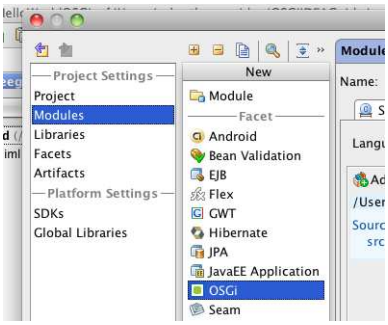
Note that you won't see the new module after closing the project settings. It is only created if there is at least one module with an OSGi facet. So you can enable that option in the template project settings and it will be enabled on any new project you create, but the module will only be created on projects were it makes sense, projects that actually do something with OSGi.

The last setting sets a default for the location of the manifest file relative to the content root of a module. So you don't need to set it for each module anew. The default is a “META-INF/MANIFEST.MF” directory directly inside the content root, but you can change it to for example “src/META-INF/MYMANIFEST.MF” to have it inside the source folder of a module or anywhere else. You will still be able to change the manifest file location for each module separately. Note that you not only define the location of the manifest file but also the name of it. Normally a manifest file is called MANIFEST.MF and Osmorc will generate MANIFEST.MF files when creating bundle jars before starting an OSGi application. You can name them however you want though if for example your build system needs them to have some other names. Osmorc recognizes files named MANIFEST.MF as manifest files and provides automatic facet detection and syntax highlighting for them.

Adding an OSGi facet to the module

Now we've prepared all that is needed to actually add OSGi functionality to our module. We open the module settings of the module by bringing up the context menu of it in the project tool window and clicking on “Module Settings”. We then click on the “+”-icon in the tool bar above the module list and choose “OSGi”. Figure 2.2 shows the last step.

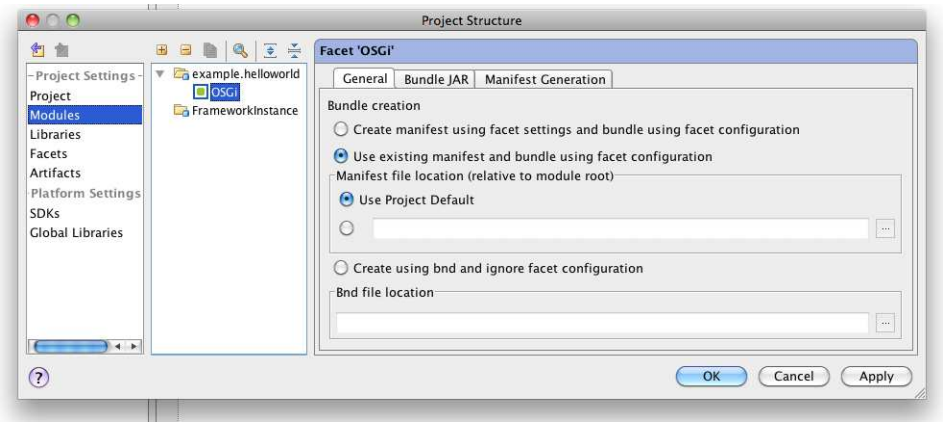
Figure 2.2. Adding an OSGi facet



The settings of the OSGi facet consists of three pages. The first one is the general settings. They are shown in Figure 2.3. On this page you define how bundles are created from IDEA modules and how Osmorc handles those modules. You can choose to generate the manifest file of the bundle from a BND file, from settings that appear on the third facet settings page titled “Manifest Generation”, or to use an existing manifest file. In this chapter we will edit the manifest file manually, so we choose “Use existing manifest and bundle using facet configuration”. Directly below that setting we can choose the location of the manifest file. We leave it on “Use Project Default”. It will choose the setting from the project settings we've seen earlier in this chapter. The predefined default is “META-INF/MANIFEST.MF” directly inside the module content root.

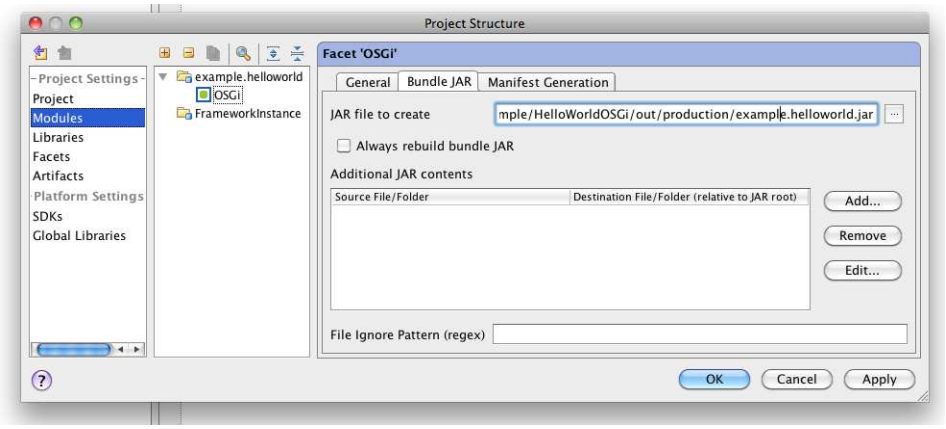
Since the manifest file does not yet exist, a warning is shown on this page and a “Create” button provides the possibility to create a manifest file for us. We click on that button.

Figure 2.3. General OSGi facet settings



On the second page of the facet settings - “Bundle JAR” (shown in Figure 2.4) - you can change where the jar for the bundle is generated into and what name it shall have. By default Osmorc generates the bundle jars into the production output folder of the project and gives them the names of their corresponding modules.

Figure 2.4. Bundle JAR OSGi facet settings



In the section “Additional JAR contents” you can add additional resources to the created bundles. This is useful if you need the bundle to contain resources that are not located in the source folder of the module and so aren't copied into the output folder when IDEA compiles the module. We'll look into this in a later chapter.

The option “Always rebuild bundle JAR” will force Osmorc to rebuild the jar before each start of an OSGi application. Normally it shouldn't be necessary to enable this option since the jar is recreated each time IDEA recompiles the bundle and Osmorc rebuilds the jar if it discovers any changes in the additional jar contents. In some contexts though those automatic detections may fail. So you can switch it on if some part of your bundle jars tends to be outdated when running an application.

The contents of the page “Manifest Generation” is amongs others the topic of Part III.

So now we're finished with the facet settings and close them with a click on “OK”.

Editing manifest files

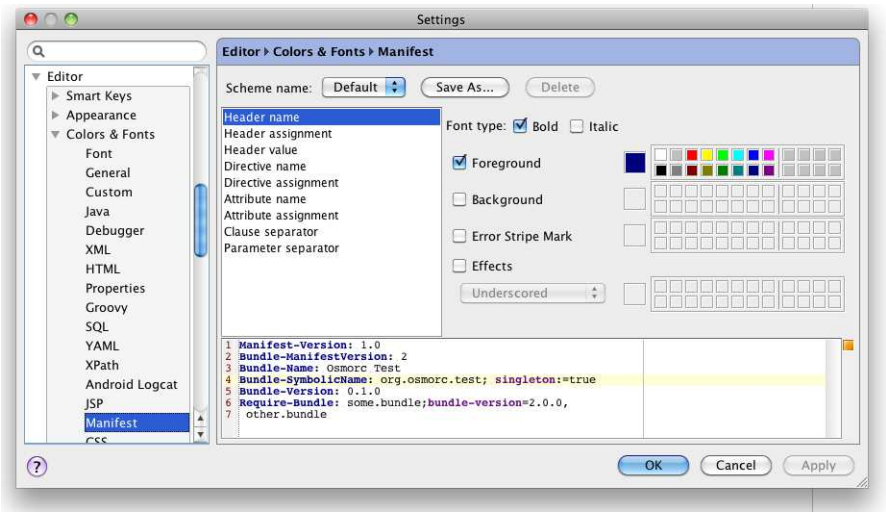
Osmorc created the folder “META-INF” in the module “example.helloworld” and inside that folder the manifest file “MANIFEST.MF”. It should look like the one shown in Example 2.1

Example 2.1. Generated manifest file of the module example.helloworld

```
Manifest-Version: 1.0.0
Bundle-ManifestVersion: 2
Bundle-Name: example.helloworld
Bundle-SymbolicName: example.helloworld
Bundle-Version: 1.0.0
```

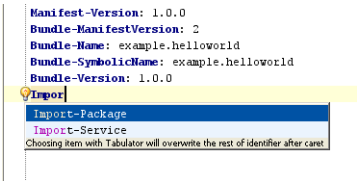
Osmorc provides syntax highlighting for manifest files. You can change the colors and fonts used for manifest files in File → Settings → IDE Settings → Editor → Colors & Fonts → Manifest as shown in Figure 2.5

Figure 2.5. Manifest file syntax hightlighting settings



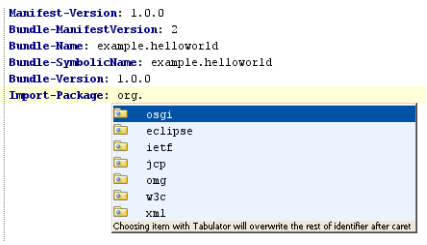
Osmorc also provides autocompletion at some points. Let's try it out by adding a new header to the manifest. Type in “Impor” in a new line at the end of the file and hit **CTRL+Space**. An autocompletion popup like the one in Figure 2.6 should appear. Choose “Import-Package”.

Figure 2.6. Manifest Header Autocompletion



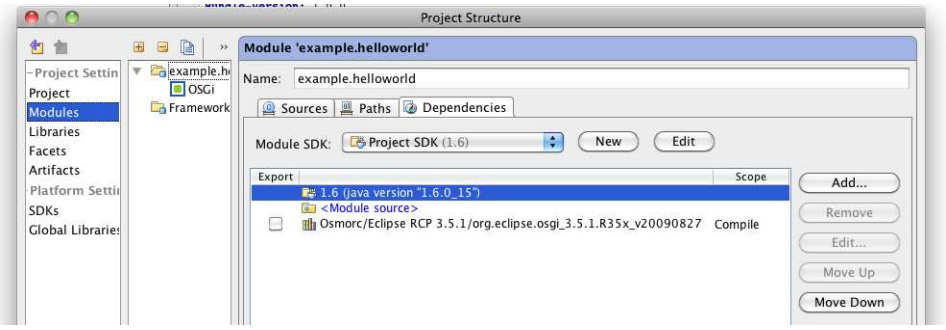
Osmorc also provides autocompletion for the packages you can import. We change the header to look like “Import-Package: org.” and again hit **CTRL+Space**. Now we get a popup looking like the one shown in Figure 2.7.

Figure 2.7. Import-Package Header Autocompletion



Use this feature to import “org.osgi.framwork”. So now we imported a package present in any framework implementing the OSGi specification. Osmorc automatically adds the necessary dependency on the corresponding global library to the module. On my computer this looks as shown in Figure 2.8. As I'm using Eclipse RCP here a bundle jar from that framework is used. Never add or remove such dependencies yourself when working with manually edited manifest files. They will be removed and added by Osmorc to reflect changes to the manifests of your bundles.

Figure 2.8. Automatically added dependency



Previous version of IDEA showed the dependencies of a module on global libraries under a node called “Libraries” inside the Project tool window. With IDEA 9 those dependencies as well as dependencies on other modules can only be seen on the “Dependencies” page of the module settings.

Implementing a BundleActivator

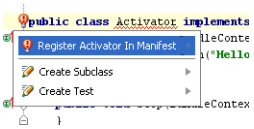
Let's get back to the goal of this chapter. We want to see a bundle activator to say “Hello World!”. We imported the package containing the “BundleActivator” interface. Now we implement that interface.

Example 2.2. A BundleActivator that says “Hello world!”

```
package example.helloworld;

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

public class Activator implements BundleActivator{
    public void start(BundleContext bundleContext) throws Exception {
        System.out.println("Hello world!");
    }

    public void stop(BundleContext bundleContext) throws Exception {
    }
}
```

If you create the BundleActivator shown in Example 2.2 you will notice that IDEA marks the name of the class “Activator” as an error. The text of the error tells us that the BundleActivator is not registered. We can fix this by moving the cursor to this name and pressing **ALT+Enter**. This will produce a popup (Figure 2.9) from which we choose “Register Activator in Manifest”. A “Bundle-Activator” header will then be added to the manifest file of the module.

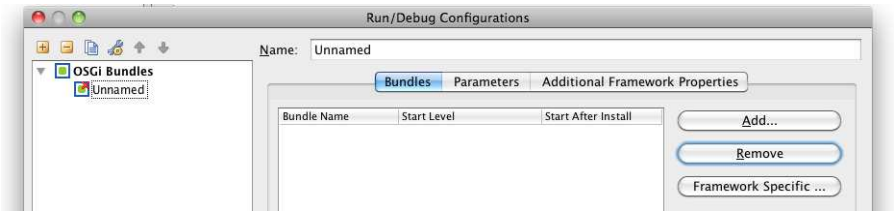
Figure 2.9. QuickFix for Activator registration



Creating and running an OSGi Run Configuration

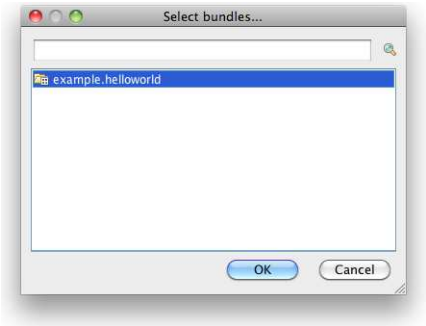
To create a run configuration for our bundle we choose “Edit Configurations” from the Run menu and click on the “+” icon in the appearing dialog. In the “Add New Configuration” list we choose “OSGi Bundles”.

Figure 2.10. Run Configuration: Bundles page



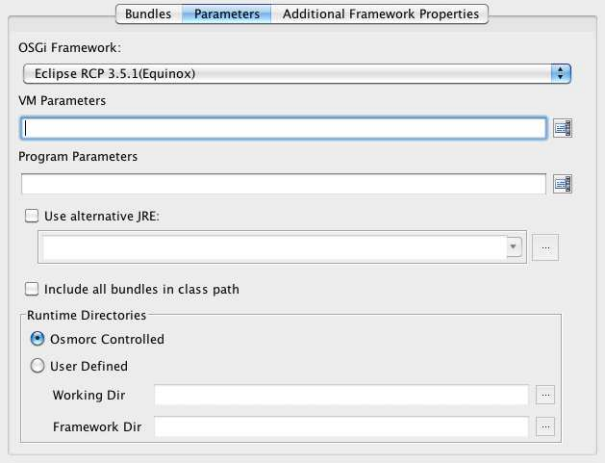
The UI for our run configuration contains three pages. The first one is labeled “Bundles” (Figure 2.10). This is where you choose which bundles are installed and whether they are started automatically. We click on the “Add” and are shown another dialog window from which we can choose the bundles to add to the run configuration. Since there currently is only one - example.helloworld - we choose that one (Figure 2.11).

Figure 2.11. Adding a bundle to the run configuration



On the parameters page (Figure 2.12) you can define parameters that are common for all framework types. The most important one is the framework instance to use for running the application. You can choose any framework instance available on your IDEA installation. It doesn't need to be the one you choose as the base for the project. You can also set VM and Program parameters just as you know it from other run configurations in IDEA.

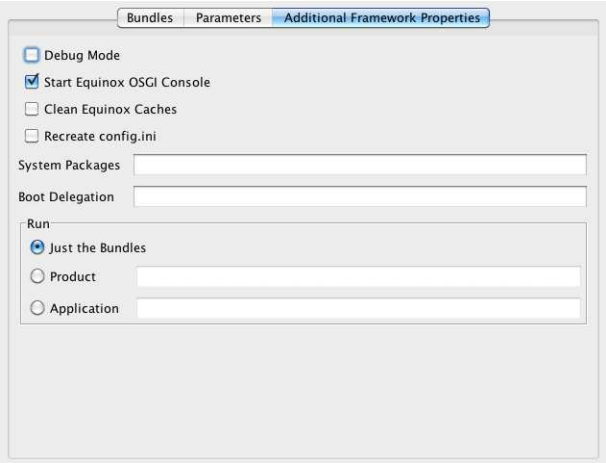
Figure 2.12. Run Configuration: Parameters page



The option “Include all bundles in class path” is only necessary in some special cases and should stay switched off by default.

Also on this page you can set the working directory and the framework directory used for starting the application. The working directory is the working directory of the VM started just like with other run configurations in IDEA. The framework directory is a directory where some frameworks put their caches into and where Osmorc generates framework specific configurations files into. Normally Osmorc will generate temporary directories and delete them after the started application is terminated. By setting the directories you can tweak the contents of the directories yourselves and add resources to them that are needed while running an application.

Figure 2.13. Run Configuration: Additional Framework Properties



On the third page (Figure 2.12) you can change settings specific to the type of framework chosen to run an application. Here we chose an Equinox instance and that's why we see Equinox specific settings.

“Clean Equinox Caches” and “Recreate config.ini” are only used when you define your own working and framework directories on the previous page. By disabling those options you can change the config.ini without Osmorc overwriting it each time and make the startup faster since the caches aren't created on each new start. But those and other settings on this page will be shown in more detail in later chapters. This time we don't need to change anything here.

Click on the “OK” button and then on the green arrow button in the tool bar of IDEA. The run configuration will start and you should see something similar to the following on the console:

Example 2.3. Console output from running the run configuration

```
osgi> Hello world!

Process finished with exit code 0
```

We reached the goal of this chapter and have seen some of the features Osmorc provides for developing OSGi applications in IDEA. The next chapter will show how to migrate an Eclipse RCP application from Eclipse as the used IDE to IDEA.

Chapter 3. Migrating an Eclipse RCP project from Eclipse to IDEA

In this chapter we'll migrate a simple Eclipse RCP project to IDEA. Eclipse RCP applications are specialized OSGi applications that build on the foundations provided by Eclipse. Naturally that kind of applications is developed with Eclipse, but with Osmorc installed in IDEA you can develop Eclipse RCP applications with IDEA. Osmorc doesn't provide any of those convenience features like building a structure for an RCP provided by Eclipse. So in this chapter we'll look at what is needed to migrate a RCP project that was created in Eclipse to enable us to work on it in IDEA.

In this chapter we assume that the reader has some knowledge about Eclipse RCP and knows how to create a RCP application in Eclipse.

Importing the Eclipse project into IDEA

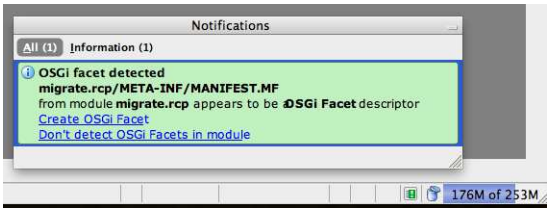
Let's say we have an Eclipse workspace named “migrateRCP” that contains an Eclipse RCP application in a plugin named “migrate.rcp”. IDEA provides support for importing Eclipse workspaces and we'll use that now. First we open the “New Project” wizard with File → New Project... and choose “Import model from external model”. On the next page we choose “Eclipse” as the source model. On the next page we specify the path to the workspace in the textfield labeled “Select Eclipse projects directory” and leave the rest at their defaults. On the last page of the wizard IDEA will show us that it recognized the plugin “migrate.rcp” as an Eclipse project and we simply click “Finish”.

IDEA will now ask us to select the used Eclipse installation. We point it to our Eclipse installation to make it happy. We don't need the global library that IDEA now generates from the plugins in that installation and will remove the dependency on it shortly.

Now we need to fix two things. To do that we open File → Project Structure and open the settings of the module. Most likely the JDK wasn't recognized correctly. We can fix that by choosing an existing JDK configuration on the dependencies page of the module settings. While we are here we also remove the dependency on the external library “ECLIPSE”. Osmorc will add dependencies on the needed plugins according to the dependencies specified in the manifest files.

Close the module settings by clicking on “OK” and notice the small green icon in the bottom toolbar of IDEA. It indicates that IDEA has detected potential facets. Clicking on it will reveal more details about the detected facets. It should look like Figure 3.1.

Figure 3.1. IDEA detects an OSGi facet



To be continued...

Part III. Generating manifests and integrating Maven

This part shows how generating manifest files and running applications using Maven or just IDEA's module system as OSGi applications is supported

Bibliography

[OSGiAlliance07] . *OSGi Service Platform Core Specification.* Release 4, Version 4.1. OSGi Alliance. Upper Saddle River, NJ. May 2007.

[McAffer06] Jeff McAffer and Jean-Michel Lemieux. *eclipse Rich Client Platform.* Designing, Coding and Packaging Java Applications. Addison-Wesley. Upper Saddle River, NJ. 2006.

[Wütherich08] Gerd Wütherich, Nils Hartmann, Bernd Kolb, and Matthias Lübken. *Die OSGi Service Platform.* Eine Einführung mit Equinox. dpunkt Verlag. Heidelberg. April 2008.