Edition 0.4

Maven
Nexus
m2eclipse

Maven

The Maven Cookbook

Sonatype

Maven

Nexus

m2eclipse

The Maven Cookbook

Tim O'Brien
Stuart McCulloch
Brian Demers

A Sonatype Open Book
Mountain View, CA

Published by:

Sonatype, Inc.
12501 Prosperity Drive
Suite 350
Silver Spring, MD 20904.

For online information and ordering of this and other Sonatype books, please visit www.sonatype.com. The publisher offers discounts on this book when ordered in quantity. For more information, please contact:
book@sonatype.com

ISBN 978-0-9842433-2-7

Editor: Tim O'Brien

List of Figures

List of Examples

Copyright

Online version published by Sonatype, Inc., 800 W. El Camino Real, Suite 400, Mountain View, CA, 94040.

Foreword: 0.4

This is an initial alpha version of the Maven Cookbook, and I mean "pre-alpha". If you are coming to the book with the expectation that this is a polished product, please check back in a few months. I'm convinced that this will be a better book because people are going to have a window into the process, you will be able to see the false starts, outline revisions, and other parts of the writing process that are usually hidden from view. You'll be able to give us feedback and reviews even before we've stated that the content is ready for review.

Tim O'Brien, Sonatype

May, 2011

Edition: 0.4

1. Changes in Edition 0.4

The following changes were introduced in Edition 0.4:

- Various changes to support the generation of a new Sonatype site.

Chapter 1. Cooking with Maven and OSGi

1.1. Introduction

OSGi is a stable, widely used framework for developing and deploying component-based systems. If you've been using Eclipse, you've already been using an OSGi container for a few years, and the pluggable and configurable nature of the Eclipse[1] platform is a product of the OSGi standards. As a concept and as an approach, OSGi is a proven standard with an installed user base of millions and growing. It might feel like an emerging standard because solid tool support is just now emerging. Prior to the last year, you might have encountered OSGi "builds", but they were usually experienced via an IDE (such as Eclipse).

Many are starting to view OSGi as the perfect solution both for deploying server-side applications and client-side GUI applications. If you need a GUI front-end or a web application to interface with a database back-end, there is a rich array of standard components to choose from in the OSGi community. Before OSGi-based servers, if you wanted a full fledged application server with a Transaction provider and JMS integration, you had to either run some large, monolithic application server like WebSphere[2], or you had to hack some custom components to a lightweight servlet-container such as Apache Tomcat[3] or Jetty[4]. Today, you wouldn't install Jetty from scratch, you will install it as an OSGi bundle in an OSGi container such as Apache Felix[5]. If you needed an embedded database like Apache Derby[6] or a transaction provider, you can now tell your container what bundles to deploy and every component is developed to operate in a standard operating environment. Once you've figured out how to instantiate an empty OSGi container such as Apache Felix there is no simpler deployment mechanism for your application, and once you've connected it to Nexus Professional[7], you've instantly solved the deployment problem in Java. This chapter takes you through the steps required to start using Maven and OSGi together and how to use Nexus Professional to support the distribution and deployment opportunities that are possible with an OSGi bundle repository (OBR).

In this chapter, we introduce some tools and techniques you can use to start developing OSGi components (or bundles) using Maven. The following recipes focus on the intersection of Apache Felix, the OPS4J project[8], and the Nexus repository manager as a bridge between Maven repositories and OSGi Bundle repositories. At the end of this chapter, you should have a clear picture of how to start developing OSGi-based applications using Maven.

1.2. Generating an OSGi Project with Maven

1.2.1. Task

You need to create a Maven multi-module project that allows you to develop a modular, OSGi-based web application.

1.2.2. Action

Use the Maven Pax Plugin[9] from OPS4J, and call the `create-project` goal. The following command-line will create a multi-module project with a groupId of `org.sonatype.mcookbook`, an artifactId of `osgi-project`, and a version of `1.0-SNAPSHOT`:

```
~/examples/osgi $ mvn org.ops4j:maven-pax-plugin:create-project \
  -DgroupId=org.sonatype.mcookbook \
  -DartifactId=osgi-project \
  -Dversion=1.0-SNAPSHOT
[INFO] Scanning for projects...
[INFO] artifact org.ops4j:maven-pax-plugin: checking for updates from central
```

[1] http://www.eclipse.org
[2] http://www-01.ibm.com/software/websphere/
[3] http://tomcat.apache.org
[4] http://www.mortbay.org/jetty/
[5] http://felix.apache.org/site/index.html
[6] http://db.apache.org/derby/
[7] www.sonatype.com/products/nexus
[8] http://www.ops4j.org/
[9] http://www.ops4j.org/projects/pax/construct/maven-pax-plugin/

```
[INFO] ------------------------------------------------------------------------
[INFO] Building Maven Default Project
[INFO]    task-segment: [org.ops4j:maven-pax-plugin:create-project] (aggregator-style)
[INFO] ------------------------------------------------------------------------
...
[INFO] [pax:create-project]
[INFO] Selecting latest archetype release within version range [1,2)
[INFO] artifact org.ops4j.pax.construct:maven-archetype-osgi-project: checking for updates from central
[INFO] ----------------------------------------------------------------------------
[INFO] Using following parameters for creating Archetype: maven-archetype-osgi-project:1.0.3
[INFO] ----------------------------------------------------------------------------
[INFO] Parameter: packageName, Value: org.sonatype.mcookbook.osgi-project
[INFO] Parameter: archetypeVersion, Value: 1.0.3
[INFO] Parameter: groupId, Value: org.sonatype.mcookbook
[INFO] Parameter: archetypeArtifactId, Value: maven-archetype-osgi-project
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] Parameter: archetypeGroupId, Value: org.ops4j.pax.construct
[INFO] Parameter: basedir, Value: ~/examples/osgi
[INFO] Parameter: package, Value: org.sonatype.mcookbook.osgi-project
[INFO] Parameter: artifactId, Value: osgi-project
[INFO] ********************* End of debug info from resources from generated POM ***********************
[INFO] Archetype created in dir: ~/examples/osgi/osgi-project
```

Once you've generated an OSGi project using the Pax Plugin, you will have the following directory structure:

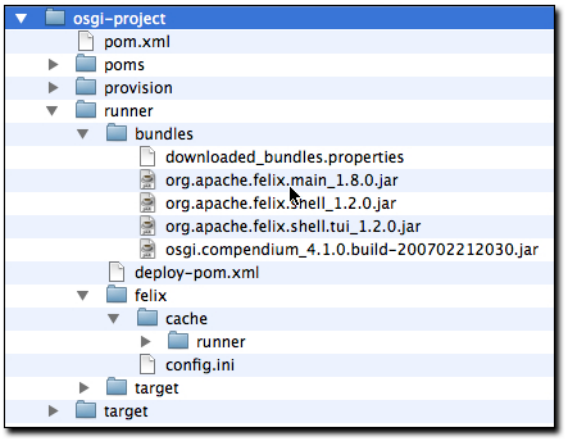

Figure 1.1. Project Structure Created by the OPS4J Pax Plugin

If you want to verify that you can build the project successfully, run **`mvn clean install`**:

```
~/examples/osgi/osgi-project $ mvn clean install
[INFO] Reactor build order:
[INFO]   org.sonatype.mcookbook.osgi-project (OSGi project)
[INFO]   osgi-project - plugin configuration
[INFO]   osgi-project - wrapper instructions
[INFO]   osgi-project - bundle instructions
[INFO]   osgi-project - imported bundles
[INFO] ------------------------------------------------------------------------
[INFO] Building org.sonatype.mcookbook.osgi-project (OSGi project)
[INFO]    task-segment: [clean, install]
[INFO] ------------------------------------------------------------------------
[INFO] [clean:clean]
[INFO] [site:attach-descriptor]
[INFO] [install:install]
[INFO] Installing ~/examples/osgi/osgi-project/target/pom-transformed.
xml to ~/.m2/repository/org/sonatype/mcookbook/osgi-project/1.0-SNAPS
HOT/osgi-project-1.0-SNAPSHOT.pom
[INFO] ------------------------------------------------------------------------
[INFO] Building osgi-project - plugin configuration
```

```
[INFO]    task-segment: [clean, install]
...
```

Note

Instead of executing these goals manually, you can also download and install the Pax-Construct scripts which can be used to automate the process of creating a new OSGi project with the Pax plugin. For more information, see the Pax Construct Quickstart page[10].

1.2.3. Detail

The generated multi-module project structure contains a parent project with the supplied groupId, artifactId, and version, and a few submodules:

osgi-project/pom.xml
: This is the parent POM

osgi-project/poms/compiled/pom.xml
: This POM serves as the parent POM to all of the compiled OSGi components you will add to this osgi-project Maven project.

osgi-project/poms/wrappers/pom.xml
: If you can't find a particular library as an OSGi component, the Pax Construct tools allow you to wrap an existing dependency artifact into an OSGi bundle. The configuration for these wrapped bundles is stored in this POM.

osgi-project/provision/pom.xml
: This POM configures the Apache Felix runtime environment that you can start by running the `pax:provision` goal. If you need to import a bundle into your runtime environment, this is the POM that contains a reference to the corresponding Maven dependency.

Note

The `osgi-project/runner` directory is not created until you run the `pax:provision` goal as shown in Section 1.3, “Starting an OSGi Container”.

1.3. Starting an OSGi Container

1.3.1. Task

You created an OSGi project, now you want to see it running in an OSGi environment. You need to start an OSGi container and load imported bundles into the runtime environment.

1.3.2. Action

First, create an OSGi project by executing the commands in Section 1.2, “Generating an OSGi Project with Maven”. Once you do this, you have a structure that has been designed to support OSGi development with Maven. To start the Apache Felix container, run **`mvn install pax:provision`** from the `osgi-project/` directory:

```
~/examples/osgi/osgi-project $ mvn install pax:provision
[INFO] Scanning for projects...
[INFO] Reactor build order:
[INFO]   org.sonatype.mcookbook.osgi-project (OSGi project)
[INFO]   osgi-project - plugin configuration
[INFO]   osgi-project - wrapper instructions
[INFO]   osgi-project - bundle instructions
[INFO]   osgi-project - imported bundles
...
[INFO] [pax:provision]
```

```
[INFO] ~~~~~~~~~~~~~~~~~~~
[INFO]  No bundles found!
[INFO] ~~~~~~~~~~~~~~~~~~~

    ______  ________  __  __
   / __  / /   __    / / / / /
  /  ___/ /  __    / _\ \ _/
 /  /    /  / /   / / _\ \
/__/    /__/ /__/ /_/ /_/

Pax Runner (1.0.0) from OPS4J - http://www.ops4j.org
----------------------------------------------------

 -> Using config [classpath:META-INF/runner.properties]
 -> Using only arguments from command line
 -> Scan bundles from [~/examples/osgi/osgi-project/runner/deploy-pom.xml]
 -> Scan bundles from [scan-pom:file:/~/examples/osgi/osgi-project/runner/deploy-pom.xml]
 -> Preparing framework [Felix 1.8.0]
 -> Downloading bundles...
 -> Using execution environment [J2SE-1.5]
 -> Runner has successfully finished his job!


Welcome to Felix.
=================

->
```

Note

Instead of executing these goals manually, you can also download and install the Pax-Construct scripts which can be used to automate the process of running Apache Felix with the Pax plugin. For more information, see the Pax Construct Quickstart page[11].

1.3.3. Detail

When you started up Felix, notice that the line directly following the execution of the `pax:provision` goal says "No bundles found!". By running `pax:provision`, you've started the Felix OSGi service platform. Felix uses the configuration from a properties file and then scans two deploy-pom.xml files for bundles that it should download and install. Finding none, it presents a simple prompt and awaits orders. At this point, we have an empty container that isn't running any custom logic or downloading any OSGi bundles.

The first time you ran `pax:provision`, the Pax plugin created a `runner` directory under `osgi-project` which captures the configuration of the runtime environment. This `runner` directory contains the following files and directories:

osgi-project/runner/deploy-pom.xml
: This POM file is generated using Pax-Construct, it contains some configuration parameters for the Apache Felix container.

osgi-project/runner/bundles
: This directory contains OSGi bundles which have been downloaded for use in Apache Felix.

osgi-project/runner/felix
: This directory contains the runtime files that are required for Apache Felix, this includes a config.ini file and a cache directory.

Once you have the console for Felix loaded, you can control the platform, load components from remote repositories, list all of the running components, and start and stop components. Try executing the command **`help`** to see a list of available commands:

```
-> help
bundlelevel <level> <id> ... | <id> - set or get bundle start level.
cd [<base-URL>]                     - change or display base URL.
exports <id> ...                    - list exported packages.
headers [<id> ...]                  - display bundle header properties.
help                                - display impl commands.
imports <id> ...                    - list imported packages.
```

```
install <URL> [<URL> ...]           - install bundle(s).
ps [-l | -s | -u]                   - list installed bundles.
refresh [<id> ...]                  - refresh packages.
requirers <id> ...                  - list requiring bundles.
requires <id> ...                   - list required bundles.
resolve [<id> ...]                  - attempt to resolve the specified bundles.
scr help                            - Declarative Services Runtime
services [-u] [-a] [<id> ...]       - list registered or used services.
shutdown                            - shutdown framework.
start [-t] <id> [<id> <URL> ...]    - start bundle(s).
startlevel [<level>]                - get or set framework start level.
stop [-t] <id> [<id> ...]           - stop bundle(s).
uninstall <id> [<id> ...]           - uninstall bundle(s).
update <id> [<URL>]                 - update bundle.
version                             - display version of framework.
```

If you execute the **`ps`** command, you can see a list of bundles with IDs that are running in the Felix container. You can start and stop bundles by running **`start`** and **`stop`** followed by the ID of the specific bundle you want to control. You can also install and uninstall bundles in the running container. Over the next few recipes we're going to fill in this picture a bit by wrapping existing JARs, importing existing OSGi bundles, and writing a custom OSGi service.

1.3.4. Resources

For more information about Apache Felix, see http://felix.apache.org/site/index.html.

1.4. Importing OSGi Bundles with Maven

1.4.1. Task

You want to configure your OSGi runtime to load bundles on startup.

1.4.2. Action

Import OSGi bundles from the Maven repository. While the Maven repository wasn't designed for OSGi like the OSGi Bundle Repositoriy (OBR) repository format, it contains a few components which contain the appropriate metadata to be referenced as OSGi components. To configure your runtime environment to load these components at runtime, you will need to invoke the pax:import-bundle goal.

As shown in the following screen listing, you are going to import several OSGi bundles from the Maven repository. First you are going to install the Apache Felix Web Management Console and then you are going to install some of its requirements. To install the first OSGi bundle run the following command:

```
~/examples/osgi/osgi-project $ mvn pax:import-bundle \
    -DgroupId=org.apache.felix \
    -DartifactId=org.apache.felix.webconsole \
    -Dversion=1.2.8
[INFO] Scanning for projects...
[INFO] Reactor build order:
[INFO]   org.sonatype.mcookbook.osgi-project (OSGi project)
[INFO]   osgi-project - plugin configuration
[INFO]   osgi-project - wrapper instructions
[INFO]   osgi-project - bundle instructions
[INFO]   osgi-project - imported bundles
[INFO] ------------------------------------------------------------------------
[INFO] Building org.sonatype.mcookbook.osgi-project (OSGi project)
[INFO]    task-segment: [pax:import-bundle] (aggregator-style)
[INFO] ------------------------------------------------------------------------
[INFO] [pax:import-bundle]
[INFO] Importing Apache Felix Web Management Console to \
       org.sonatype.mcookbook.osgi-project.build:provision:pom:1.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------
```

```
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 6 seconds
[INFO] Finished at: Mon Jul 13 18:30:37 CDT 2009
[INFO] Final Memory: 8M/16M
[INFO] ------------------------------------------------------------------------
```

After installing the Apache Felix Web Management console, you will need to install some of its prerequisites. The following commands install version 2.1 of the Java Servlet API, the Jetty HTTP server, and the `org.apache.felix.scr` component. As you can see, some of the OSGi bundle you are installing are simple libraries and APIs, while others are complex servers.

```
~/examples/osgi/osgi-project $ mvn pax:import-bundle \
    -DgroupId=org.apache.felix \
    -DartifactId=javax.servlet
    -Dversion=1.0.0
~/examples/osgi/osgi-project $ mvn pax:import-bundle \
    -DgroupId=org.apache.felix \
    -DartifactId=org.apache.felix.scr \
    -Dversion=1.0.8
~/examples/osgi/osgi-project $ mvn pax:import-bundle \
    -DgroupId=org.apache.felix \
    -DartifactId=org.apache.felix.http.jetty \
    -Dversion=1.0.1
```

Once you have bundled these OSGi bundles, you can start your runtme environment. Because the `pax:import-bundle` goal changes some of the POMs in your `osgi-project`, you will need to run **`mvn install`** before you can run **`mvn pax:provision`**. After importing bundles, they will appear in `provision/pom.xml`.

```
~/examples/osgi/osgi-project $ mvn install pax:provision
[INFO] [pax:provision]
[INFO] Installing ~/examples/osgi/osgi-project/runner/target/pom-transformed.xml
       to ~/.m2/repository/org/sonatype/mcookbook/osgi-project/build/deployment/1.0-SNAPSHOT/\
          deployment-1.0-SNAPSHOT.pom
    ______  ________  __  __
   / __  / /   __   / / / / /
  /  ___/ /  __    / _\ \ _/
 /  /    /  / /   / / _\ \
/__/    /__/ /__/ /_/ /_/

Pax Runner (1.0.0) from OPS4J - http://www.ops4j.org
-----------------------------------------------------

 -> Using config [classpath:META-INF/runner.properties]
 -> Using only arguments from command line
 -> Scan bundles from [~/examples/osgi/osgi-project/runner/deploy-pom.xml]
 -> Scan bundles from [scan-pom:file:~/examples/osgi/osgi-project/runner/deploy-pom.xml]
 -> Provision bundle [mvn:org.apache.felix/org.apache.felix.webconsole/1.2.8,
at default start level, bundle will be started, bundle will be loaded from the cache]
 -> Provision bundle [mvn:org.apache.felix/javax.servlet/1.0.0, at default start
level, bundle will be started, bundle will be loaded from the cache]
 -> Provision bundle [mvn:org.apache.felix/org.apache.felix.http.jetty/1.0.1,
at default start level, bundle will be started, bundle will be loaded from the cache]
 -> Provision bundle [mvn:org.apache.felix/org.apache.felix.scr/1.0.8, at default
start level, bundle will be started, bundle will be loaded from the cache]
 -> Preparing framework [Felix 1.8.0]
 -> Downloading bundles...
 -> mvn:org.apache.felix/org.apache.felix.webconsole/1.2.8 : 102399 bytes
 -> mvn:org.apache.felix/javax.servlet/1.0.0 : 30450 bytes @ [ 10150kBps ]
 -> mvn:org.apache.felix/org.apache.felix.http.jetty/1.0.1 : 102399 bytes @
 -> mvn:org.apache.felix/org.apache.felix.scr/1.0.8 : 112058 bytes @ [ 8619kBps ]
 -> Using execution environment [J2SE-1.5]
 -> Runner has successfully finished his job!
```

```
Welcome to Felix.
=================

-> org.mortbay.log:Logging to org.mortbay.log via org.apache.felix.http.jetty.LogServiceLog
org.mortbay.log:Init SecureRandom.
org.mortbay.log:started org.mortbay.jetty.servlet.HashSessionIdManager@34a33d
org.mortbay.log:started org.mortbay.jetty.servlet.HashSessionManager@46ac5a
org.mortbay.log:starting OsgiServletHandler@dd7786
org.mortbay.log:started OsgiServletHandler@dd7786
org.mortbay.log:starting SessionHandler@d23e75
org.mortbay.log:started SessionHandler@d23e75
org.mortbay.log:starting org.mortbay.jetty.servlet.Context@9deddb{/,null}
org.mortbay.log:starting ErrorHandler@28bda
org.mortbay.log:started ErrorHandler@28bda
org.mortbay.log:started org.mortbay.jetty.servlet.Context@9deddb{/,null}
org.mortbay.log:jetty-6.1.x
org.mortbay.log:started Realm[OSGi HTTP Service Realm]==[]
org.mortbay.log:started org.mortbay.thread.QueuedThreadPool@f13b08
org.mortbay.log:starting Server@5a936b
org.mortbay.log:started org.mortbay.jetty.nio.SelectChannelConnector$1@9576c3
org.mortbay.log:Started SelectChannelConnector@0.0.0.0:8080
org.mortbay.log:started SelectChannelConnector@0.0.0.0:8080
org.mortbay.log:started Server@5a936b
org.mortbay.log:started /system/console/*
org.mortbay.log:started /system/console/res
```

After running mvn pax:provision, you have an instance of Felix that is running the Apache Felix Web Management Console application. Fire up a web brower, and go to http://localhost:8080/system/console to load the interface. The default administrative username and password is `admin/admin`.

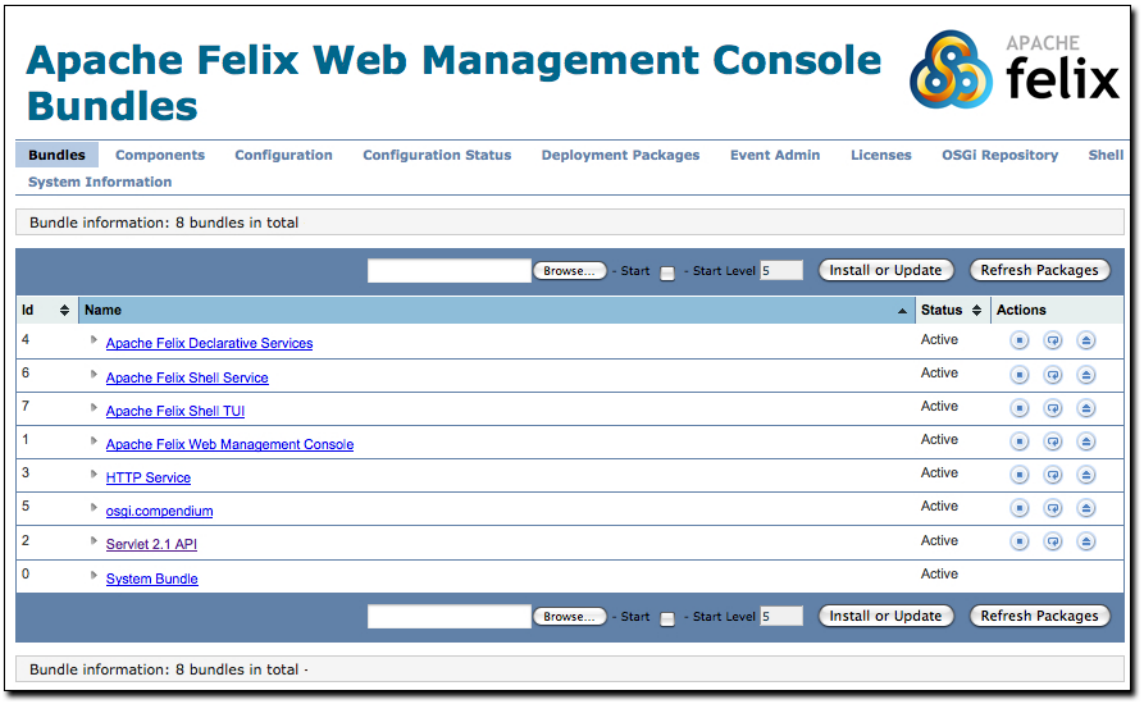

Figure 1.2. Apache Felix Web Management Console

Note

Instead of executing these goals manually, you can also download and install the Pax-Construct scripts which can be used to automate the process of importing bundles with the Pax plugin. For more information, see the Pax Construct Quickstart page[12].

1.4.3. Detail

Once you start the web management console for Apache Felix, you can start to manage the installed bundles. You can start and stop bundles via the interface, and you can install bundles from remote repositories. If you prefer to use the shell interface from the previous recipe, click on the Shell interface and you can enter in any of the commands you used at the command-line Felix management console as shown in Figure 1.3, “Running the Apache Felix Shell via the Administrative Web Console”.

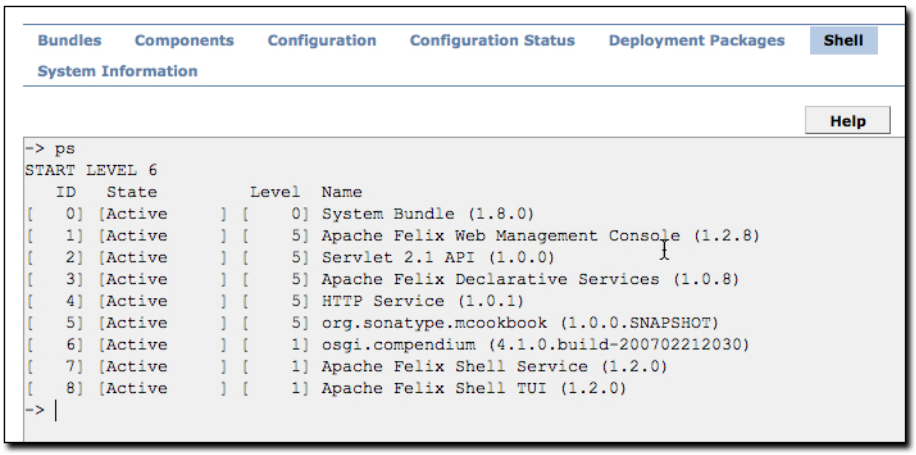

Figure 1.3. Running the Apache Felix Shell via the Administrative Web Console

You can manage Felix interactions with events and repositories, and you can also restart, stop, and change the default run levels. Clicking on the "System Information" tab will also allow you to get some statistics about the runtime container as shown in Figure 1.4, “Managing the Apache Felix via System Information”.

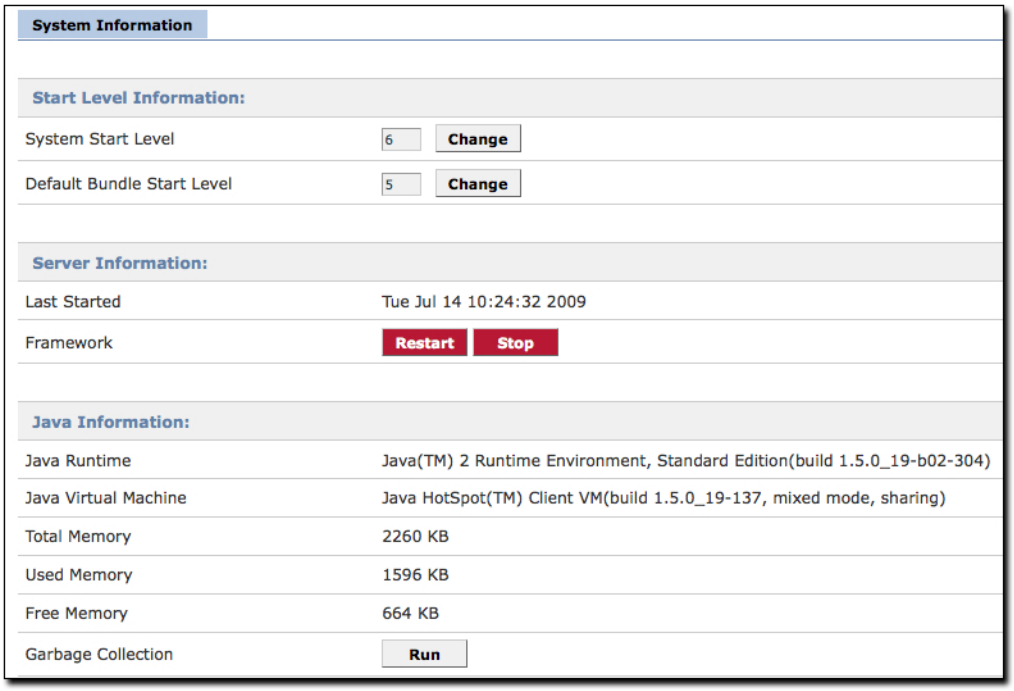

Figure 1.4. Managing the Apache Felix via System Information

When we imported these bundles into the project, they were added to the `provision/pom.xml` file, and we had to install all of a bundle's dependencies one by one. Instead of listing out each of bundle's dependencies on the command-line, we can tell the Pax

plugin to import transitive bundle dependencies and optional bundles. The following command line will instruct the Pax plugin to install a bundle's full compile and runtime dependencies in `provision/pom.xml`.

```
~/examples/osgi/osgi-project $ mvn pax:import-bundle \
                                        -DgroupId=org.apache.felix \
                                        -DartifactId=org.apache.felix.http.jetty \
                                        -Dversion=1.0.1 \
                                        -DimportTransitive=true \
                                        -DimportOptional=true \
                                        -DwidenScope=true
```

The `importTransitive` and `importOptional` options tell the Pax plugin to look at the OBR metadata to find bundle dependencies. The `widenScope` parameter tells the Pax plugin to install all compile and runtime dependencies as bundles in `provision/pom.xml`.

1.4.4. Resources

For more information about the Apache Felix Web Console, see http://cwiki.apache.org/confluence/display/FELIX/Apache+Felix+Web+Console.

1.5. Creating an OSGi Bundle with Maven

1.5.1. Task

You need to create you own OSGi bundle project to integrate with the runtime environment created in the previous recipes.

1.5.2. Action

Run the `pax:create-bundle` goal. The following screen listing runs the `pax:create-bundle` to create a new Maven project for an OSGi bundle with a package org.sonatype.mcookbook and a name `osgi-bundle`. Run this command from the `~/examples/osgi/osgi-project` directory:

```
~/examples/osgi/osgi-project $ mvn pax:create-bundle \
  -Dpackage=org.sonatype.mcookbook \
  -Dname=osgi-bundle \
  -Dversion=1.0-SNAPSHOT
```

Once this command has been completed, run **`mvn install pax:provision`** and then verify that the component is present by running **`ps`** at the Felix administrative console:

```
~/examples/osgi/osgi-project $ mvn install pax:provision
...
[INFO] Reactor build order:
[INFO]   org.sonatype.mcookbook.osgi-project (OSGi project)
[INFO]   osgi-project - plugin configuration
[INFO]   osgi-project - wrapper instructions
[INFO]   osgi-project - bundle instructions
[INFO]   osgi-project - imported bundles
[INFO]   org.sonatype.mcookbook
..
[INFO] ------------------------------------------------------------------------
[INFO] Building org.sonatype.mcookbook
[INFO]    task-segment: [install]
[INFO] ------------------------------------------------------------------------
[INFO] [resources:resources]
[WARNING] Using platform encoding (MacRoman actually) to copy filtered resources, i.e. build is platform \
          dependent!
[INFO] Copying 1 resource
[INFO] Copying 0 resource
```

```
[INFO] [pax:compile]
[INFO] Nothing to compile - all classes are up to date
[INFO] [pax:testCompile]
[INFO] No sources to compile
...
[INFO] [install:install]
[INFO] Installing ~/examples/osgi/osgi-project/org.sonatype.mcookbook/target/\
       org.sonatype.mcookbook-1.0-SNAPSHOT.jar to \
       ~/.m2/repository/.m2/repository/org/sonatype/mcookbook/osgi-project/org.sonatype.mcookbook/\
       1.0-SNAPSHOT/org.sonatype.mcookbook-1.0-SNAPSHOT.jar
[INFO] [bundle:install]
[INFO] Parsing file:~/.m2/repository/.m2/repository/repository.xml
[INFO] Installing org/sonatype/mcookbook/osgi-project/org.sonatype.mcookbook/1.0-SNAPSHOT/\
       org.sonatype.mcookbook-1.0-SNAPSHOT.jar
[INFO] Writing OBR metadata
[INFO] ------------------------------------------------------------------------
[INFO] Building org.sonatype.mcookbook.osgi-project (OSGi project)
[INFO]    task-segment: [pax:provision] (aggregator-style)
[INFO] ------------------------------------------------------------------------
```

The Pax plugin bundles the new OSGi bundle into a JAR file which is deployed to the local Maven repository in ~/.m2/repository. The resulting artifact has the following identifiers:

- groupId: `org.sonatype.mcookbook.osgi-project`
- artifactId: `org.sonatype.mcookbook`
- version: `1.0-SNAPSHOT`

Continuing on with this particular execution of Maven, lines corresponding to this new custom OSGi bundle have been highlighted. The `mvn:org.sonatype.mcookbook.osgi-project/org.sonatype.mcookbook/1.0-SNAPSHOT` is provisioned to Apache Felix and after running the **`ps`** command we see that the `org.sonatype.mcookbook` bundle is deployed with ID 5.

```
[INFO] [pax:provision]
[INFO] Installing ~/examples/osgi/osgi-project/runner/target/pom-transformed.xml to \
       ~/.m2/repository/.m2/repository/org/sonatype/mcookbook/osgi-project/build/deployment/1.0-SNAPSHOT/\
       deployment-1.0-SNAPSHOT.pom

Pax Runner (1.0.0) from OPS4J - http://www.ops4j.org
-----------------------------------------------------

 -> Using config [classpath:META-INF/runner.properties]
 -> Using only arguments from command line
 -> Scan bundles from [~/examples/osgi/osgi-project/runner/deploy-pom.xml]
 -> Scan bundles from [scan-pom:file:~/examples/osgi/osgi-project/runner/deploy-pom.xml]
 -> Provision bundle [mvn:org.apache.felix/org.apache.felix.webconsole/1.2.8, at default start level, \
    bundle will be started, bundle will be loaded from the cache]
 -> Provision bundle [mvn:org.apache.felix/javax.servlet/1.0.0, at default start level, bundle will be \
    started, bundle will be loaded from the cache]
 -> Provision bundle [mvn:org.apache.felix/org.apache.felix.scr/1.0.8, at default start level, bundle will be \
    started, bundle will be loaded from the cache]
 -> Provision bundle [mvn:org.apache.felix/org.apache.felix.http.jetty/1.0.1, at default start level, bundle \
    will be started, bundle will be loaded from the cache]
 -> Provision bundle [mvn:org.sonatype.mcookbook.osgi-project/org.sonatype.mcookbook/1.0-SNAPSHOT, at default \
    start level, bundle will be started, bundle will be loaded from the cache]
 -> Preparing framework [Felix 1.8.0]
...
 -> Runner has successfully finished his job!


Welcome to Felix.
=================

-> org.mortbay.log:Logging to org.mortbay.log via org.apache.felix.http.jetty.LogServiceLog
STARTING org.sonatype.mcookbook
REGISTER org.sonatype.mcookbook.ExampleService
```

```
org.mortbay.log:Init SecureRandom.
...
org.mortbay.log:started /system/console/res

-> ps
START LEVEL 6
   ID   State         Level  Name
[   0] [Active     ] [    0] System Bundle (1.8.0)
[   1] [Active     ] [    5] Apache Felix Web Management Console (1.2.8)
[   2] [Active     ] [    5] Servlet 2.1 API (1.0.0)
[   3] [Active     ] [    5] Apache Felix Declarative Services (1.0.8)
[   4] [Active     ] [    5] HTTP Service (1.0.1)
[   5] [Active     ] [    5] org.sonatype.mcookbook (1.0.0.SNAPSHOT)
[   6] [Active     ] [    1] osgi.compendium (4.1.0.build-200702212030)
[   7] [Active     ] [    1] Apache Felix Shell Service (1.2.0)
[   8] [Active     ] [    1] Apache Felix Shell TUI (1.2.0)
```

Note

Instead of executing these goals manually, you can also download and install the Pax-Construct scripts. For more information, see the Pax Construct Quickstart page[13].

1.5.3. Detail

So you just deployed your own OSGi bundle to a OSGi runtime environment, but where is the code? And, what is in this OSGi bundle anyway? Open up your osgi-project directory and you'll see a directory named org.sonatype.mcookbook/ with a directory structure similar to that shown in Figure 1.5, “Custom Bundle org.sonatype.mcookbook”.

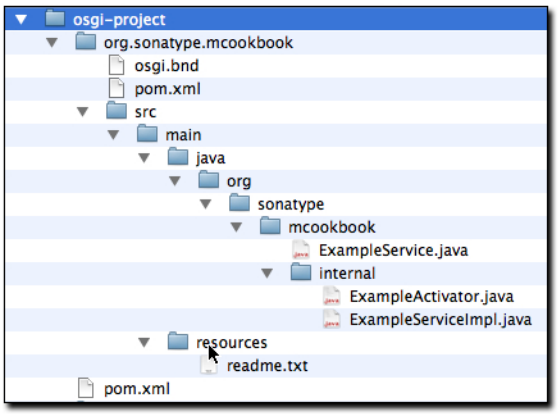

Figure 1.5. Custom Bundle org.sonatype.mcookbook

This sample project defines a single interface named `ExampleService` which offers a single method `scramble()`.

Example 1.1. ExampleService Interface

```
package org.sonatype.mcookbook;

/**
 * Public API representing an example OSGi service
 */
public interface ExampleService
{
    // public methods go here...

    String scramble( String text );
}
```

This `ExampleService` interface is implemented by the class `ExampleServiceImpl` shown in the following program listing.

Example 1.2. ExampleServiceImpl Implementation of the ExampleService Interface

```
package org.sonatype.mcookbook.internal;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

import org.sonatype.mcookbook.ExampleService;

/**
 * Internal implementation of our example OSGi service
 */
public final class ExampleServiceImpl
    implements ExampleService
{
    // implementation methods go here...

    public String scramble( String text )
    {
        List charList = new ArrayList();

        char[] textChars = text.toCharArray();
        for( int i = 0; i < textChars.length; i++ )
        {
            charList.add( new Character( textChars[i] ) );
        }

        Collections.shuffle( charList );

        char[] mixedChars = new char[text.length()];
        for( int i = 0; i < mixedChars.length; i++ )
        {
            mixedChars[i] = ( (Character) charList.get( i ) ).charValue();
        }

        return new String( mixedChars );
    }
}
```

Notice how neither `ExampleService` nor `ExampleServiceImpl` have any knowledge of the OSGi environment. For the `ExampleService` to be made available to the OSGi environment, there needs to be a `BundleActivator` which can instantiate the service and interact with the OSGi environment. This is included in the project in the form of an `ExampleActivator` which has a method named `start()` which takes a `BundleContext` instance and which calls `registerService()` on this context with an instance of `ExampleServiceImpl`.

Example 1.3. The BundleActivator implementation: ExampleActivator

```
package org.sonatype.mcookbook.internal;

import java.util.Dictionary;
import java.util.Properties;

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

import org.sonatype.mcookbook.ExampleService;

/**
 * Extension of the default OSGi bundle activator
 */
public final class ExampleActivator
```

```
    implements BundleActivator
{
    /**
     * Called whenever the OSGi framework starts our bundle
     */
    public void start( BundleContext bc )
        throws Exception
    {
        System.out.println( "STARTING org.sonatype.mcookbook" );

        Dictionary props = new Properties();
        // add specific service properties here...

        System.out.println( "REGISTER org.sonatype.mcookbook.ExampleService" );

        // Register our example service implementation in the OSGi service registry
        bc.registerService( ExampleService.class.getName(), new ExampleServiceImpl(), props );
    }

    /**
     * Called whenever the OSGi framework stops our bundle
     */
    public void stop( BundleContext bc )
        throws Exception
    {
        System.out.println( "STOPPING org.sonatype.mcookbook" );

        // no need to unregister our service - the OSGi framework handles it for us
    }
}
```

Lastly, the `osgi.bnd` file in `~/examples/osgi/osgi-project/org.sonatype.mcookbook` supplies the bundle configuration which identifies the `ExampleActivator` class as the appropriate `Bundle-Activator`.

Example 1.4. osgi.bnd Bundle Configuration

```
#----------------------------------------------------------------
# Use this file to add customized Bnd instructions for the bundle
#----------------------------------------------------------------

Bundle-Activator: ${bundle.namespace}.internal.ExampleActivator
```

1.6. Starting an Alternative OSGi Framework (Knopflerfish)

1.6.1. Task

You want to run your OSGi bundles in the Knopflerfish OSGi runtime environment.

1.6.2. Action

Pass in the framework configuration parameter when running the `pax:provision` goal. To start Knopflerfish, run **`mvn pax:provision -Dframework=kf`**.

```
~/examples/osgi/osgi-project $ mvn pax:provision -Dframework=kf
[INFO] Scanning for projects...
...
[INFO] [pax:provision]
[INFO] Installing ~/examples/osgi/osgi-project/runner/target/pom-transformed.xml to \
       /Users/Tim/.m2/repository/org/sonatype/mcookbook/osgi-project/build/deployment/1.0-SNAPSHOT/\
       deployment-1.0-SNAPSHOT.pom
[INFO] artifact org.ops4j.pax.runner:pax-runner: checking for updates from central
Downloading: http://repo1.maven.org/maven2/org/ops4j/pax/runner/pax-runner/1.1.1/pax-runner-1.1.1.jar
1067K downloaded  (pax-runner-1.1.1.jar)
```

```
    ______  ________  __  __
   / __  / /  __   / / / / /
  /  ___/ /  __    / _\ \ _/
 /  /    /  / /   / / _\ \
/__/    /__/ /__/ /_/ /_/

Pax Runner (1.1.1) from OPS4J - http://www.ops4j.org
----------------------------------------------------

 -> Using config [classpath:META-INF/runner.properties]
 -> Using only arguments from command line
 -> Scan bundles from [~/examples/osgi/osgi-project/runner/deploy-pom.xml]
 -> Scan bundles from [scan-pom:file:~/examples/osgi/osgi-project/runner/deploy-pom.xml]
...
 -> Preparing framework [Knopflerfish 2.3.1]
 -> Downloading bundles...
 -> Knopflerfish 2.3.1 : 366660 bytes @ [ 160kBps ]
 -> org.osgi.compendium : 689150 bytes @ [ 256kBps ]
 -> Knopflerfish Console : 36329 bytes @ [ 55kBps ]
 -> Knopflerfish Console TTY : 6153 bytes @ [ 1538kBps ]
 -> Knopflerfish Framework Commands : 26090 bytes @ [ 158kBps ]
 -> Using execution environment [J2SE-1.5]
 -> Runner has successfully finished his job!

Knopflerfish OSGi framework, version 4.1.3
Copyright 2003-2009 Knopflerfish. All Rights Reserved.

See http://www.knopflerfish.org for more information.
Loading xargs url file:knopflerfish/config.ini
Installed and started: file:bundles/osgi.compendium_4.0.1.jar (id#1)
Installed and started: file:bundles/org.knopflerfish.bundle.console_2.0.1.jar (id#2)
Installed and started: file:bundles/org.knopflerfish.bundle.consoletty-IMPL_2.0.0.jar (id#3)
Installed and started: file:bundles/org.knopflerfish.bundle.frameworkcommands-IMPL_2.0.5.jar (id#4)
Installed and started: file:bundles/org.apache.felix.webconsole_1.2.8.jar (id#5)
Installed and started: file:bundles/org.apache.felix.javax.servlet_1.0.0.jar (id#6)
Installed and started: file:bundles/org.apache.felix.scr_1.0.8.jar (id#7)
Installed and started: file:bundles/org.apache.felix.http.jetty_1.0.1.jar (id#8)
Installed and started: file:bundles/org.sonatype.mcookbook_1.0.0.SNAPSHOT.jar (id#9)
> STARTING org.sonatype.mcookbook
REGISTER org.sonatype.mcookbook.ExampleService
org.mortbay.log:Logging to org.mortbay.log via org.apache.felix.http.jetty.LogServiceLog
Framework launched
org.mortbay.log:Init SecureRandom.
org.mortbay.log:started org.mortbay.jetty.servlet.HashSessionIdManager@b890dc
org.mortbay.log:started org.mortbay.jetty.servlet.HashSessionManager@6321e6
org.mortbay.log:starting OsgiServletHandler@4683c0
org.mortbay.log:started OsgiServletHandler@4683c0
org.mortbay.log:starting SessionHandler@73bc22
org.mortbay.log:started SessionHandler@73bc22
org.mortbay.log:starting org.mortbay.jetty.servlet.Context@a0d346{/,null}
org.mortbay.log:starting ErrorHandler@449afc
org.mortbay.log:started ErrorHandler@449afc
org.mortbay.log:started org.mortbay.jetty.servlet.Context@a0d346{/,null}
org.mortbay.log:jetty-6.1.x
org.mortbay.log:started Realm[OSGi HTTP Service Realm]==[]
org.mortbay.log:started org.mortbay.thread.QueuedThreadPool@39491b
org.mortbay.log:starting Server@eea7f0
org.mortbay.log:started org.mortbay.jetty.nio.SelectChannelConnector$1@31db04
org.mortbay.log:Started SelectChannelConnector@0.0.0.0:8080
org.mortbay.log:started SelectChannelConnector@0.0.0.0:8080
org.mortbay.log:started Server@eea7f0
org.mortbay.log:started /system/console/*
org.mortbay.log:started /system/console/res
```

The output of this last command is going to be somewhat confusing. Even though we're starting the application using Knopflerfish, we're still deploying some OSGi components from Apache Felix, and we're deploying the Apache Felix Web Management Console. The interesting thing about OSGi: it all still works. The Apache Felix Web Management Console can be used to manage standard

OSGi component even when it is running in the Knopflerfish container. To test this, go to http://localhost:8080/system/console, and log in with the default admin/admin credentials. Click on the configuration status tab, and then scroll down to see properties that prove that your Apache Felix web management console is running atop Knopflerfish.

1.6.3. Detail

For more information about Knopflerfish, see http://www.knopflerfish.org. For more information about the frameworks available to the Pax plugin, see http://paxrunner.ops4j.org/space/Pax+Runner.

1.7. Starting an Alternative OSGi Framework (Equinox)

1.7.1. Task

You want to run your OSGi bundles in Eclipse's Equinox OSGi runtime environment.

1.7.2. Action

Pass in the framework configuration parameter when running the `pax:provision` goal. To start Knopflerfish, run **`mvn pax:provision -Dframework=equinox`**.

```
~/examples/osgi/osgi-project $ mvn pax:provision -Dframework=equinox
[INFO] Scanning for projects...
[INFO] Reactor build order:
[INFO]   org.sonatype.mcookbook.osgi-project (OSGi project)
[INFO]   osgi-project - plugin configuration
[INFO]   osgi-project - wrapper instructions
[INFO]   osgi-project - bundle instructions
[INFO]   osgi-project - imported bundles
[INFO]   org.sonatype.mcookbook
[INFO] ------------------------------------------------------------------------
[INFO] Building org.sonatype.mcookbook.osgi-project (OSGi project)
[INFO]    task-segment: [pax:provision] (aggregator-style)
[INFO] ------------------------------------------------------------------------
[INFO] [pax:provision]
[INFO] Installing /Users/Tim/Library/Code/sonatype/maven-cookbook/mcookbook-examples/osgi/osgi-project/\
       runner/target/pom-transformed.xml to \
       /Users/Tim/.m2/repository/org/sonatype/mcookbook/osgi-project/build/deployment/1.0-SNAPSHOT/\
       deployment-1.0-SNAPSHOT.pom
    ______  ________  __  __
   / __  / /  __   / / / / /
  /  ___/ /  __   / _\ \ _/
 /  /    /  / /  / / _\ \
/__/    /__/ /__/ /_/ /_/

Pax Runner (1.1.1) from OPS4J - http://www.ops4j.org
-----------------------------------------------------

 -> Using config [classpath:META-INF/runner.properties]
 -> Using only arguments from command line
 -> Scan bundles from [/Users/Tim/Library/Code/sonatype/maven-cookbook/mcookbook-examples/osgi/osgi-project/\
                       runner/deploy-pom.xml]
 -> Scan bundles from [scan-pom:file:/Users/Tim/Library/Code/sonatype/maven-cookbook/mcookbook-examples/osgi/\
                       osgi-project/runner/deploy-pom.xml]
 -> Provision bundle [mvn:org.apache.felix/org.apache.felix.webconsole/1.2.8, at default start level, bundle \
    will be started, bundle will be loaded from the cache]
 -> Provision bundle [mvn:org.apache.felix/javax.servlet/1.0.0, at default start level, bundle will be \
    started, bundle will be loaded from the cache]
 -> Provision bundle [mvn:org.apache.felix/org.apache.felix.scr/1.0.8, at default start level, bundle will be \
    started, bundle will be loaded from the cache]
 -> Provision bundle [mvn:org.apache.felix/org.apache.felix.http.jetty/1.0.1, at default start level, bundle \
    will be started, bundle will be loaded from the cache]
 -> Provision bundle [mvn:org.sonatype.mcookbook.osgi-project/org.sonatype.mcookbook/1.0-SNAPSHOT, at default \
    start level, bundle will be started, bundle will be loaded from the cache]
```

```
 -> Preparing framework [Equinox 3.5.0]
 -> Downloading bundles...
 -> Equinox 3.5.0 : 1122602 bytes @ [ 300kBps ]
 -> Eclipse utilities : 22471 bytes @ [ 64kBps ]
 -> Eclipse compendium services : 66065 bytes @ [ 113kBps ]
 -> Using execution environment [J2SE-1.5]
 -> Runner has successfully finished his job!


osgi> STARTING org.sonatype.mcookbook
REGISTER org.sonatype.mcookbook.ExampleService
org.mortbay.log:Logging to org.mortbay.log via org.apache.felix.http.jetty.LogServiceLog
org.mortbay.log:Init SecureRandom.
org.mortbay.log:started org.mortbay.jetty.servlet.HashSessionIdManager@9ed5d6
org.mortbay.log:started org.mortbay.jetty.servlet.HashSessionManager@fd245f
org.mortbay.log:starting OsgiServletHandler@5f00f9
org.mortbay.log:started OsgiServletHandler@5f00f9
org.mortbay.log:starting SessionHandler@38ef1d
org.mortbay.log:started SessionHandler@38ef1d
org.mortbay.log:starting org.mortbay.jetty.servlet.Context@cc3baa{/,null}
org.mortbay.log:starting ErrorHandler@da6d09
org.mortbay.log:started ErrorHandler@da6d09
org.mortbay.log:started org.mortbay.jetty.servlet.Context@cc3baa{/,null}
org.mortbay.log:jetty-6.1.x
org.mortbay.log:started Realm[OSGi HTTP Service Realm]==[]
org.mortbay.log:started org.mortbay.thread.QueuedThreadPool@f2e41d
org.mortbay.log:starting Server@5e4dbe
org.mortbay.log:started org.mortbay.jetty.nio.SelectChannelConnector$1@9a44d6
org.mortbay.log:Started SelectChannelConnector@0.0.0.0:8080
org.mortbay.log:started SelectChannelConnector@0.0.0.0:8080
org.mortbay.log:started Server@5e4dbe
org.mortbay.log:started /system/console/*
org.mortbay.log:started /system/console/res
```

Again, the output of this last command is going to be somewhat confusing. Even though we're starting the application using Equinox, we're still deploying some OSGi components from Apache Felix, and we're deploying the Apache Felix Web Management Console. While the Felix Web Management Console can also run in the Equinox OSGi runtime environment, take note that there are some missing classes toward the end of the previous console output. This is likely due to the fact that there is a missing bundle or some Felix-specific functionality that is present in the Felix management console. To test the console, go to http://localhost:8080/system/console, and log in with the default `admin`/`admin` credentials.

1.7.3. Detail

For more information about Equinox, see http://www.eclipse.org/equinox/. For more information about the frameworks available to the Pax plugin, see http://paxrunner.ops4j.org/space/Pax+Runner.

1.8. Deploying OSGi Bundles to a Maven Repository

1.8.1. Task

You need to configure your project to deploy your own custom OSGi bundles to a Maven repository.

1.8.2. Action

The solution is to download an install Nexus Professional, and configure your Maven environment to deploy snapshot and release artifacts to a Hosted Maven repository. To install Nexus Professional, complete the following steps:

1. Download Nexus Professional from http://www.sonatype.com/products/downloads.

2. Unpack the Nexus Professional archive on your local workstation, or on the machine that will host Nexus Professional for your workgroup or organization.

3. Start Nexus Professional by running the startup script in **`${nexus_install_dir}/bin/jsw/<platform>/nexus start`** where <platform> is replaced with the appropriate platform-specific directory.

The following commands will install and start Nexus Professional 1.3.5:

```
~/programs $ cp ~/Downloads/nexus-professional-webapp-1.3.5-bundle.tar.gz ~/programs
~/programs $ tar xvzf nexus-professional-webapp-1.3.5-bundle.tar.gz
~/programs $ cd nexus-professional-webapp-1.3.5
~/programs/nexus-professional-webapp-1.3.5 $ ./bin/jsw/macosx-universal-32/nexus start
Starting Sonatype Nexus Repository Manager...
Started Sonatype Nexus Repository Manager.
```

Create a `~/.m2/settings.xml` and copy the XML from Example 1.5, "settings.xml to configure deployment credentials and mirrors for Nexus" into this new Maven Settings XML file.

Example 1.5. settings.xml to configure deployment credentials and mirrors for Nexus

```
<settings>
  <mirrors>
    <mirror>
      <!--This sends everything else to /public -->
      <id>nexus</id>
      <mirrorOf>*</mirrorOf>
      <url>http://localhost:8081/nexus/content/groups/public</url>
    </mirror>
  </mirrors>
  <profiles>
    <profile>
      <id>nexus</id>
      <!--Enable snapshots for the built in central repo to direct -->
      <!--all requests to nexus via the mirror -->
      <repositories>
        <repository>
          <id>central</id>
          <url>http://central</url>
          <releases><enabled>true</enabled></releases>
          <snapshots><enabled>true</enabled></snapshots>
        </repository>
      </repositories>
     <pluginRepositories>
        <pluginRepository>
          <id>central</id>
          <url>http://central</url>
          <releases><enabled>true</enabled></releases>
          <snapshots><enabled>true</enabled></snapshots>
        </pluginRepository>
      </pluginRepositories>
    </profile>
  </profiles>
  <servers>
    <server>
      <id>nx-snapshots</id>
      <username>deployment</username>
      <password>deployment123</password>
    </server>
    <server>
      <id>nx-releases</id>
      <username>deployment</username>
      <password>deployment123</password>
    </server>
  </servers>
  <activeProfiles>
    <!--make the profile active all the time -->
    <activeProfile>nexus</activeProfile>
  </activeProfiles>
</settings>
```

Now, go into the `osgi-project` example project from the previous labs and add the XML from Example 1.6, "Distribution Management Settings for the osgi-project Project" into the top-level parent in `~/examples/osgi/osgi-project/pom.xml`. You can add the `distributionManagement` element as a child of the `project` element after all of the existing children.

Example 1.6. Distribution Management Settings for the osgi-project Project

```
<project>
  ...
  <distributionManagement>
    <repository>
      <id>nx-releases</id>
      <name>Nexus Releases</name>
      <url>http://localhost:8081/nexus/content/repositories/releases</url>
    </repository>
    <snapshotRepository>
      <id>nx-snapshots</id>
      <name>Nexus Snapshots</name>
      <url>http://localhost:8081/nexus/content/repositories/snapshots</url>
    </snapshotRepository>
  </distributionManagement>
</project>
```

Now, deploy the bundle from your osgi-project to the Nexus Snapshots repository by running mvn deploy -DremoteOBR.

```
~/osgi/osgi-project $ mvn deploy -DremoteOBR
[INFO] Scanning for projects...
...
[INFO] [deploy:deploy]
[INFO] Retrieving previous build number from nx-snapshots
Uploading: http://localhost:8081/nexus/content/repositories/snapshots/org/sonatype/mcookbook/osgi-project/\
           1.0-SNAPSHOT/osgi-project-1.0-20090715.030835-2.pom
2K uploaded  (osgi-project-1.0-20090715.030835-2.pom)
[INFO] Retrieving previous metadata from nx-snapshots
[INFO] Uploading repository metadata for: 'artifact org.sonatype.mcookbook:osgi-project'
[INFO] Retrieving previous metadata from nx-snapshots
[INFO] Uploading repository metadata for: 'snapshot org.sonatype.mcookbook:osgi-project:1.0-SNAPSHOT'
...
[INFO] [deploy:deploy]
[INFO] Retrieving previous build number from nx-snapshots
Uploading: http://localhost:8081/nexus/content/repositories/snapshots/org/sonatype/mcookbook/osgi-project/\
           build/shared-plugin-settings/1.0-SNAPSHOT/shared-plugin-settings-1.0-20090715.030835-2.pom
2K uploaded  (shared-plugin-settings-1.0-20090715.030835-2.pom)
[INFO] Retrieving previous metadata from nx-snapshots
[INFO] Uploading repository metadata for: 'artifact \
       org.sonatype.mcookbook.osgi-project.build:shared-plugin-settings'
[INFO] Retrieving previous metadata from nx-snapshots
[INFO] Uploading repository metadata for: 'snapshot \
       org.sonatype.mcookbook.osgi-project.build:shared-plugin-settings:1.0-SNAPSHOT'
...
[INFO] [deploy:deploy]
[INFO] Retrieving previous build number from nx-snapshots
Uploading: http://localhost:8081/nexus/content/repositories/snapshots/org/sonatype/mcookbook/osgi-project/\
           build/wrapper-bundle-settings/1.0-SNAPSHOT/wrapper-bundle-settings-1.0-20090715.030835-2.pom
1K uploaded  (wrapper-bundle-settings-1.0-20090715.030835-2.pom)
[INFO] Retrieving previous metadata from nx-snapshots
[INFO] Uploading repository metadata for: 'artifact \
       org.sonatype.mcookbook.osgi-project.build:wrapper-bundle-settings'
[INFO] Retrieving previous metadata from nx-snapshots
[INFO] Uploading repository metadata for: 'snapshot \
       org.sonatype.mcookbook.osgi-project.build:wrapper-bundle-settings:1.0-SNAPSHOT'
...
[INFO] [deploy:deploy]
[INFO] Retrieving previous build number from nx-snapshots
...
[INFO] [bundle:deploy]
[INFO] LOCK http://localhost:8081/nexus/content/repositories/snapshots/repository.xml
```

```
[INFO] Downloading repository.xml
[INFO] Parsing file:/var/folders/qR/qRpEDfQcFPmRZpsXgPHOok+++TM/-Tmp-/1247627320963708149070778155179 4.xml
[INFO] Deploying org/sonatype/mcookbook/osgi-project/org.sonatype.mcookbook/1.0-SNAPSHOT/\
       org.sonatype.mcookbook-1.0-20090715.030835-2.jar
[INFO] Writing OBR metadata
[INFO] Uploading repository.xml
[INFO] UNLOCK http://localhost:8081/nexus/content/repositories/snapshots/repository.xml
```

To summarize, you downloaded and installed Nexus Professional, configured your Maven Settings to supply the default credentials for the deployment user, you added the snapshots and releases repositories to your project's pom.xml, and then you deployed your project's SNAPSHOT artifacts to he snapshot repository and updated an OBR-compatible repository.xml file also stored in the Nexus-hosted Maven repository.

Warning

We skipped some extremely important setup tasks for Nexus Professional, namely changing all of the default passwords. This example uses the default username and password for the deployment user (deployment/deployment123). To change this password, log into the Nexus interface by going to http://localhost:8081/nexus, logging in with the default administrative credentials (admin/admin123), clicking on Users under Security in the left navigation menu, and then right-clicking on the deployment user in the list of users. Once you right-click on the deployment user, select "Set Password" and then supply a new password. Don't leave Nexus unprotected with the default username and password.

1.8.3. Detail

To verify that your artifacts made it into the Snapshots repository on your local Nexus instance, go to the Nexus interface at http://localhost:8081/nexus and login as the administrative user. The default administrative credentials are the username "admin" and the password "admin123". Once you login as the admin user, click on Repositories in the left navigation menu, and then select the Snapshots repository. If you browse the contents of the repository you should see contents similar to the contents shown in Figure 1.6, “The Contents of the Snapshots Repository after Deployment”.

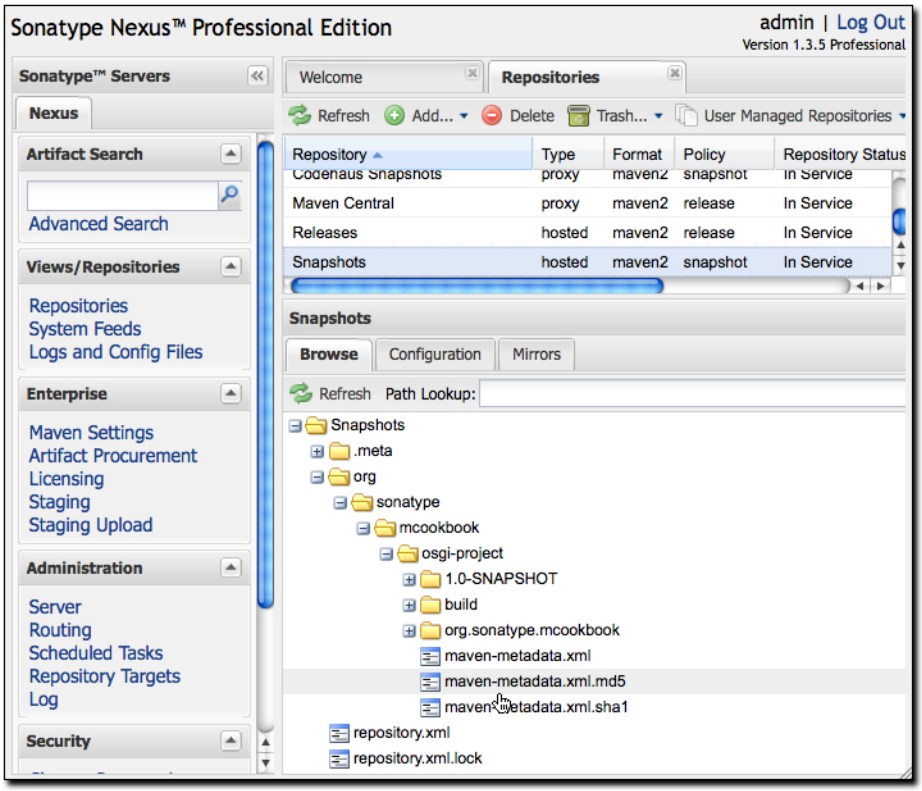

Figure 1.6. The Contents of the Snapshots Repository after Deployment

You can see from Figure 1.6, “The Contents of the Snapshots Repository after Deployment” that the repository contains the `org.sonatype.mcookbook` OSGi bundle under the `org.sonatype.mcookbook.osgi-project` groupId and that the Pax plugin took care of creating an OSGi bundle repository XML file at `repository.xml`.

If you want to test how easy it would be to install this component directly from Nexus:

1. Download Apache Felix from http://www.apache.org/dist/felix/felix-1.8.0.tar.gz.
2. Unpack the Felix distribution on your local workstation.
3. Change directory to the Felix directory.
4. Start Apache Felix with **`java -jar ./bin/felix.jar`**
5. Add the Snapshots repository as an OBR repository by copying the URL to the repository.xml and passing it to the obr command: **`obr add-url http://localhost:8081/nexus/content/repositories/snapshots/repository.xml`**
6. List the contents of the OBR repository.
7. Deploy the `org.sonatype.mcookbook` bundle.
8. List the installed bundles.

These commands are captured in the following screen listing:

```
~/programs $ wget http://www.apache.org/dist/felix/felix-1.8.0.tar.gz
--2009-07-14 22:46:19--  http://www.apache.org/dist/felix/felix-1.8.0.tar.gz
Resolving www.apache.org... 140.211.11.130
Connecting to www.apache.org|140.211.11.130|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 901620 (880K) [application/x-gzip]
Saving to: `felix-1.8.0.tar.gz'
2009-07-14 22:46:20 (924 KB/s) - `felix-1.8.0.tar.gz' saved [901620/901620]

~/programs $ tar xvzf felix-1.8.0.tar.gz
felix-1.8.0/
felix-1.8.0/bin/
felix-1.8.0/bin/felix.jar
...
felix-1.8.0/NOTICE
~/programs $ cd felix-1.8.0
~/programs/felix-1.8.0 $ java -jar ./bin/felix.jar

Welcome to Felix.
=================

-> obr add-url http://localhost:8081/nexus/content/repositories/snapshots/repository.xml
-> obr list
org.sonatype.mcookbook (1.0.0.SNAPSHOT)
-> obr deploy org.sonatype.mcookbook
Target resource(s):
-------------------
   org.sonatype.mcookbook (1.0.0.SNAPSHOT)

Deploying...done.
-> ps
START LEVEL 1
   ID   State         Level  Name
[   0] [Active     ] [    0] System Bundle (1.8.0)
[   1] [Active     ] [    1] Apache Felix Shell Service (1.2.0)
[   2] [Active     ] [    1] Apache Felix Shell TUI (1.2.0)
[   3] [Active     ] [    1] Apache Felix Bundle Repository (1.4.0)
[   4] [Installed  ] [    1] org.sonatype.mcookbook (1.0.0.SNAPSHOT)
-> start 4
```

```
STARTING org.sonatype.mcookbook
REGISTER org.sonatype.mcookbook.ExampleService
->
```

In summary, you've deployed an OSGi bundle to a Maven repository using the Pax plugin from OPS4J and you've successfully deployed it to an independent instance of Apache Felix directly from your Nexus instance. This is a powerful demonstration because it provides a model for how deployment to production of staging can work once you've standardized on a Maven repository manager and OSGi as a deployment framework.

Note

You can also setup and install Nexus Open Source, and rely on the first-class support that the the Pax plugin provides for OSGi bundles stored in a standard Maven repository. In the context of OSGi development, the main difference between Nexus Professional and Nexus Open Source is that Nexus Professional provides direct support for OBR repositories, allowing you to proxy, host, and group OSGi bundle repositories and convert existing Maven repositories to OSGi bundle repositories.

1.9. Transforming a Maven Repository into an OSGi Bundle Repository

In the previous recipe, you published an OSGi bundle to a Maven repository, and you relied on the Pax plugin to read and write the repository.xml file. While this approximates an OBR repository, you'd like Nexus to take care of generating the OBR metadata from the repository contents, and you don't want to rely on individual builds to lock, read, parse, alter, write, and unlock a shared OBR metadata file. Instead of relying on the Pax plugin, this recipe shows you how to configure an OBR repository in Nexus Professional.

1.9.1. Task

You need to expose OSGi bundles in a Maven repository using the OBR repository format.

1.9.2. Action

Use Nexus Professional, and create a virtual OBR repository which will scan a Maven repository for OSGi bundle artifacts and generate the OBR XML metadata in response to a change in the Maven repository. Building upon the example from the previous recipe, create a Virtual OBR repository named Snapshots OBR which uses the Snapshots repository as a source. To do this:

1. Load the Nexus interface in a web browser by opening the URL http://localhost:8081/nexus.
2. Login as an administrative user using the default credentials of admin/admin123 (if you haven't already changed the default password).
3. Click on Repositories in the left navigation menu.
4. Click on the Add.. button above the list of Nexus repositories and groups.
5. Select "Virtual Repository" from the resulting dropdown.
6. In the New Virtual Repository window, supply the following values as shown in Figure 1.7, “Creating a New Virtual OBR Repository”:
   a. Repository ID: **`snapshots-obr`**
   b. Repository Name: **`Snapshots OBR`**
   c. Provider: **`OBR`**
   d. Source Nexus Repository ID: **`snapshots`**
   e. Synchronize on Startup: **`False`**

7. Click the Save button to create the new Virtual repository.

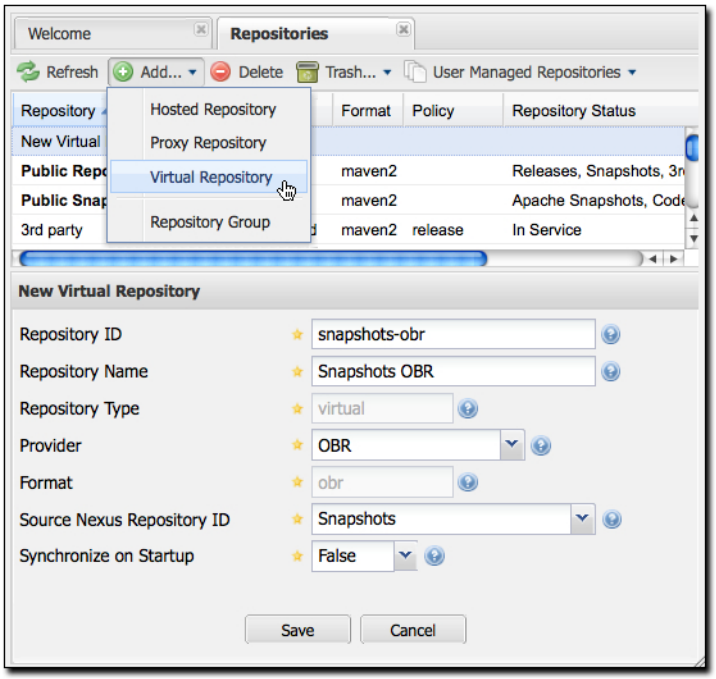

Figure 1.7. Creating a New Virtual OBR Repository

1.9.3. Detail

To verify that our new Virtual OBR repository is generating the the appropriate XML based on the contents of the Maven repository, click on Repositories in the left navigation menu, and then select "Snapshots OBR" from the list of repositories. If you look at the Browse tab for this repository and drill into the `.meta/` directory, you can then right-click on the `obr.xml` file and select Download from the context menu. This should load the repository XML in a browser, and the contents of this XML file should resemble the XML file in Example 1.7, “Metadata for the Snapshots OBR Virtual Repository”. The URL for this new repository XML should be http://localhost:8081/nexus/content/repositories/snapshots-obr/.meta/obr.xml.

Example 1.7. Metadata for the Snapshots OBR Virtual Repository

```
<repository name='Snapshots OBR' lastmodified='1247630990695'>
<resource id='org.sonatype.mcookbook/1.0.0.SNAPSHOT' presentationname='org.sonatype.mcookbook'
          symbolicname='org.sonatype.mcookbook'
          uri='../org/sonatype/mcookbook/osgi-project/org.sonatype.mcookbook/1.0-SNAPSHOT/\
               org.sonatype.mcookbook-1.0-20090715.030813-1.jar' version='1.0.0.SNAPSHOT'>
  <description>
    Generated using Pax-Construct
  </description>
  <size>
    5345
  </size>
  <capability name='bundle'>
    <p n='manifestversion' v='2'/>

    <p n='presentationname' v='org.sonatype.mcookbook'/>
    <p n='symbolicname' v='org.sonatype.mcookbook'/>
    <p n='version' t='version' v='1.0.0.SNAPSHOT'/>
  </capability>
  <capability name='package'>
    <p n='package' v='org.sonatype.mcookbook'/>
    <p n='version' t='version' v='1.0.0.SNAPSHOT'/>
  </capability>
  <require extend='false' filter='(&amp;(package=org.osgi.framework)(version&gt;=0.0.0))' multiple='false'
```

```
          name='package' optional='false'>

    Import package org.osgi.framework
  </require>
  <require extend='false' filter='(&amp;(package=org.sonatype.mcookbook)(version&gt;=1.0.0.SNAPSHOT))'
          multiple='false' name='package' optional='false'>
    Import package org.sonatype.mcookbook ;version=1.0.0.SNAPSHOT
  </require>
</resource>
<resource id='org.sonatype.mcookbook/1.0.0.SNAPSHOT' presentationname='org.sonatype.mcookbook'
          symbolicname='org.sonatype.mcookbook'
          uri='../org/sonatype/mcookbook/osgi-project/org.sonatype.mcookbook/1.0-SNAPSHOT/\
               org.sonatype.mcookbook-1.0-20090715.030835-2.jar' version='1.0.0.SNAPSHOT'>
  <description>
    Generated using Pax-Construct
  </description>
  <size>

    5346
  </size>
  <capability name='bundle'>
    <p n='manifestversion' v='2'/>
    <p n='presentationname' v='org.sonatype.mcookbook'/>
    <p n='symbolicname' v='org.sonatype.mcookbook'/>
    <p n='version' t='version' v='1.0.0.SNAPSHOT'/>
  </capability>
  <capability name='package'>

    <p n='package' v='org.sonatype.mcookbook'/>
    <p n='version' t='version' v='1.0.0.SNAPSHOT'/>
  </capability>
  <require extend='false' filter='(&amp;(package=org.osgi.framework)(version&gt;=0.0.0))' multiple='false'
          name='package' optional='false'>
    Import package org.osgi.framework
  </require>
  <require extend='false' filter='(&amp;(package=org.sonatype.mcookbook)(version&gt;=1.0.0.SNAPSHOT))'
          multiple='false' name='package' optional='false'>
    Import package org.sonatype.mcookbook ;version=1.0.0.SNAPSHOT
  </require>
</resource>

</repository>
```

If you want to test how easy it would be to install this component directly from this new Nexus virtual OBR directory. If you've already installed Felix, from the previous recipe, delete the felix-1.8.0 directory and recreate the entire environment to take note of any changes in the behavior of Felix when using the new Snapshots OBR virtual repository:

1. Download Apache Felix from http://www.apache.org/dist/felix/felix-1.8.0.tar.gz.
2. Unpack the Felix distribution on your local workstation.
3. Change directory to the Felix directory.
4. Start Apache Felix with **`java -jar ./bin/felix.jar`**
5. Add the Snapshots repository as an OBR repository by copying the URL to the repository.xml and passing it to the obr command: **`obr add-url http://localhost:8081/nexus/content/repositories/snapshots-obr/.meta/obr.xml`**
6. List the contents of the OBR repository.
7. Deploy the `org.sonatype.mcookbook` bundle.
8. List the installed bundles.

These commands are captured in the following screen listing:

```
~/programs $ wget http://www.apache.org/dist/felix/felix-1.8.0.tar.gz
--2009-07-14 22:46:19--  http://www.apache.org/dist/felix/felix-1.8.0.tar.gz
Resolving www.apache.org... 140.211.11.130
Connecting to www.apache.org|140.211.11.130|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 901620 (880K) [application/x-gzip]
Saving to: `felix-1.8.0.tar.gz'
2009-07-14 22:46:20 (924 KB/s) - `felix-1.8.0.tar.gz' saved [901620/901620]

~/programs $ tar xvzf felix-1.8.0.tar.gz
felix-1.8.0/
felix-1.8.0/bin/
felix-1.8.0/bin/felix.jar
...
felix-1.8.0/NOTICE
~/programs $ cd felix-1.8.0
~/programs/felix-1.8.0 $ java -jar ./bin/felix.jar

Welcome to Felix.
=================

-> obr add-url http://localhost:8081/nexus/content/repositories/snapshots-obr/.meta/obr.xml
-> obr list
org.sonatype.mcookbook (1.0.0.SNAPSHOT, ...)
-> obr deploy org.sonatype.mcookbook
Target resource(s):
-------------------
   org.sonatype.mcookbook (1.0.0.SNAPSHOT)

Deploying...done.
-> ps
START LEVEL 1
   ID   State         Level  Name
[   0] [Active     ] [    0] System Bundle (1.8.0)
[   1] [Active     ] [    1] Apache Felix Shell Service (1.2.0)
[   2] [Active     ] [    1] Apache Felix Shell TUI (1.2.0)
[   3] [Active     ] [    1] Apache Felix Bundle Repository (1.4.0)
[   4] [Installed  ] [    1] org.sonatype.mcookbook (1.0.0.SNAPSHOT)
-> start 4
STARTING org.sonatype.mcookbook
REGISTER org.sonatype.mcookbook.ExampleService
->
```

Aside from the output of the **`obr list`** command, this process was exactly the same as the previous recipe. The main difference between this recipe and the previous recipe is that we do not rely on the Pax Plugin to generate the repository XML file that makes the repository an OBR repository. Instead, we rely upon the OBR integration of Nexus Professional. Every time the underlying source repository (in this case the hosted snapshots repository) is changed, Nexus Professional will regenerate the OBR XML in this virtual OBR repository. Using this virtual repository, you can simply deploy your project's OSGi and non-OSGi artifacts to the same Maven repository and then you can expose OSGi bundles to systems such as Felix using Nexus Professional's support for virtual OBR repositories.

If you are using Nexus Professional's OBR support and exposing a Maven repository as an OBR repository using a virtual repository, there is no need for you to pass the **`-DremoteOBR`** option to mvn deploy as we did in the previous recipe.

1.10. Proxying OSGi Bundle Repositories

If you've followed the recipes in this chapter sequentially, you deployed an artifact to a Maven repository and you've seen two ways to get that repository to an OSGi container: via a standard Maven repository and via a virtual OBR repository that acts as a transformer between Maven and OSGi. This chapter has focused on hosted repositories so far, but one of the most common reasons for people to use repository managers is to proxy remote repositories. In this recipe, you will learn how to proxy a remote OSGi bundle repository using Nexus Professional's support for OBR.

1.10.1. Task

You would like to maintain a local cache of a remote OSGi Bundle repository that is populated with artifacts as they are requested. You need to do this for a few reasons, but mostly because you realized how inefficient it is for everyone in your organization to continuously query a remote server and download artifacts over the public internet. You are looking for more stability and control over what people are using from a remote OSGi bundle repository.

1.10.2. Action

Use Nexus Professional to create a proxy repository for a remote OSGi bundle repository. To do this:

1. Load the Nexus interface in a web browser by opening the URL http://localhost:8081/nexus.
2. Login as an administrative user using the default credentials of admin/admin123 (if you haven't already changed the default password).
3. Click on Repositories in the left navigation menu.
4. Click on the Add.. button above the list of Nexus repositories and groups.
5. Select "Proxy Repository" from the resulting dropdown.
6. In the New Virtual Repository window, supply the following values as shown in Figure 1.8, “Creating a New Proxy Repository”:
   a. Repository ID: **`felix-obr`**
   b. Repository Name: **`Felix OBR`**
   c. Provider: **`OBR`**
   d. Remote Storage Location: **`http://felix.apache.org/obr/releases.xml`**
7. Click the Save button to create the new Proxy repository.

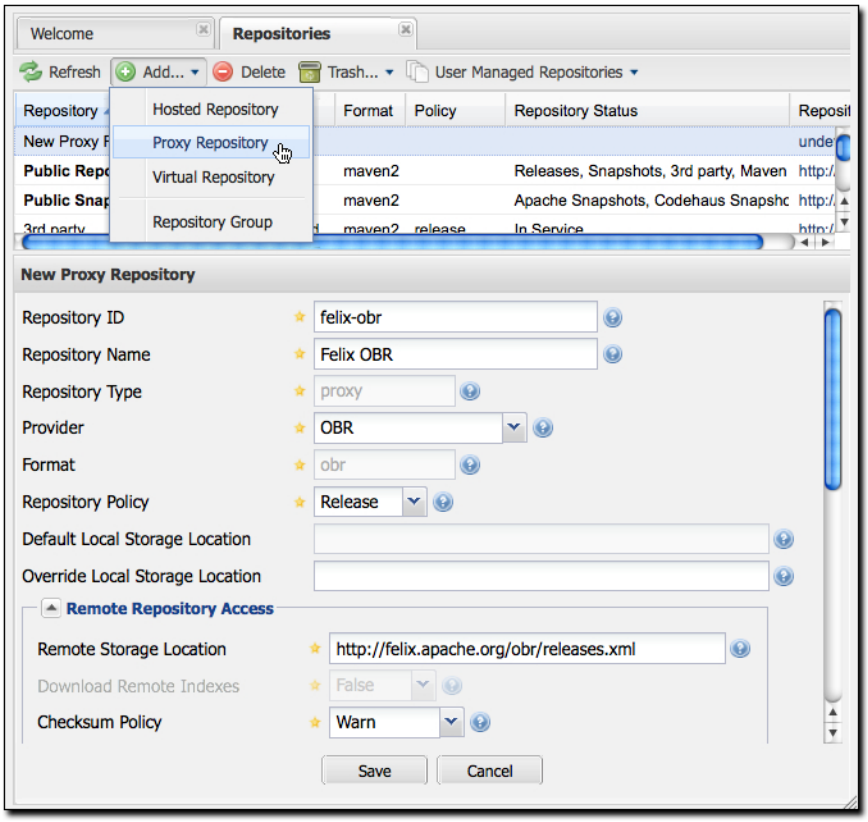

Figure 1.8. Creating a New Proxy Repository

1.10.3. Detail

To verify that our new Proxy OBR repository is generating the the appropriate XML based on the remote OBR repository, click on Repositories in the left navigation menu, and then select "Felix OBR" from the list of repositories. If you look at the Browse tab for this repository and drill into the `.meta/` directory, you can then right-click on the `obr.xml` file and select Download from the context menu. The URL for this new repository XML should be http://localhost:8081/nexus/content/repositories/felix-obr/.meta/obr.xml. Be warned, this is going to be a very large XML file, and you might not be able to make sense of the document by reading the source directly. Instead of dealing with this unwieldy XML file, use Apache Felix to browse the contents.

If you want to see the contents of this new proxy repository for the Felix OBR repository, install Felix and add a reference to the Nexus proxy repository:

1. Download Apache Felix from http://www.apache.org/dist/felix/felix-1.8.0.tar.gz.
2. Unpack the Felix distribution on your local workstation.
3. Change directory to the Felix directory.
4. Start Apache Felix with **`java -jar ./bin/felix.jar`**
5. Add the Snapshots repository as an OBR repository by copying the URL to the repository.xml and passing it to the obr command: **`obr add-url http://localhost:8081/nexus/content/repositories/felix-obr/.meta/obr.xml`**
6. List the contents of the OBR repository.
7. Deploy the Servlet 2.1 bundle.
8. Deploy the Apache Felix Web Management Console bundle.

9. Start the installed bundles.

These commands are captured in the following screen listing:

```
~/programs $ wget http://www.apache.org/dist/felix/felix-1.8.0.tar.gz
--2009-07-14 22:46:19--  http://www.apache.org/dist/felix/felix-1.8.0.tar.gz
Resolving www.apache.org... 140.211.11.130
Connecting to www.apache.org|140.211.11.130|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 901620 (880K) [application/x-gzip]
Saving to: `felix-1.8.0.tar.gz'
2009-07-14 22:46:20 (924 KB/s) - `felix-1.8.0.tar.gz' saved [901620/901620]

~/programs $ tar xvzf felix-1.8.0.tar.gz
felix-1.8.0/
felix-1.8.0/bin/
felix-1.8.0/bin/felix.jar
...
felix-1.8.0/NOTICE
~/programs $ cd felix-1.8.0
~/programs/felix-1.8.0 $ java -jar ./bin/felix.jar

Welcome to Felix.
=================

-> obr add-url http://localhost:8081/nexus/content/repositories/felix-obr/.meta/obr.xml
-> obr list
Apache Felix Bundle Repository (1.2.1, ...)
Apache Felix Configuration Admin Service (1.0.4, ...)
Apache Felix Declarative Services (1.0.8, ...)
Apache Felix EventAdmin (1.0.0)
Apache Felix File Install (1.2.0, ...)
Apache Felix HTTP Service Jetty (1.0.1, ...)
Apache Felix iPOJO (1.2.0, ...)
Apache Felix iPOJO (0.8.0)
Apache Felix iPOJO Arch Command (1.2.0, ...)
Apache Felix iPOJO Composite (1.2.0)
Apache Felix iPOJO Composite (1.0.0, ...)
Apache Felix iPOJO Event Admin Handler (1.2.0, ...)
Apache Felix iPOJO Extender Pattern Handler (1.2.0)
Apache Felix iPOJO Extender Pattern Handler (1.0.0, ...)
Apache Felix iPOJO JMX Handler (1.2.0, ...)
Apache Felix iPOJO Temporal Service Dependency Handler (1.2.0, ...)
Apache Felix iPOJO White Board Pattern Handler (1.2.0, ...)
Apache Felix Log Service (1.0.0)
Apache Felix Metatype Service (1.0.2, ...)
Apache Felix Prefrences Service (1.0.2)
Apache Felix Remote Shell (1.0.2)
Apache Felix Shell Service (1.0.2, ...)
Apache Felix Shell TUI (1.0.2, ...)
Apache Felix UPnP Base Driver (0.8.0)
Apache Felix UPnP Extra (0.4.0)
Apache Felix UPnP Tester (0.4.0)
Apache Felix Web Management Console (1.2.10, ...)
OSGi OBR Service API (1.0.0)
OSGi R4 Compendium Bundle (4.0.0)
Servlet 2.1 API (1.0.0)
-> obr deploy "Servlet 2.1 API"
Target resource(s):
-------------------
   Servlet 2.1 API (1.0.0)

Deploying...done.
-> obr deploy "Apache Felix Web Management Console"
Target resource(s):
-------------------
   Apache Felix Web Management Console (1.2.10)
```

```
Required resource(s):
--------------------
   Apache Felix Log Service (1.0.0)
   Apache Felix Declarative Services (1.0.8)
   Apache Felix HTTP Service Jetty (1.0.1)
   OSGi R4 Compendium Bundle (4.0.0)

Deploying...done.
-> ps
START LEVEL 1
   ID   State         Level  Name
[   0] [Active     ] [    0] System Bundle (1.8.0)
[   1] [Active     ] [    1] Apache Felix Shell Service (1.2.0)
[   2] [Active     ] [    1] Apache Felix Shell TUI (1.2.0)
[   3] [Active     ] [    1] Apache Felix Bundle Repository (1.4.0)
[   4] [Installed  ] [    1] Servlet 2.1 API (1.0.0)
[   5] [Installed  ] [    1] Apache Felix Log Service (1.0.0)
[   6] [Installed  ] [    1] Apache Felix Declarative Services (1.0.8)
[   7] [Installed  ] [    1] HTTP Service (1.0.1)
[   8] [Installed  ] [    1] Apache Felix Web Management Console (1.2.10)
[   9] [Installed  ] [    1] OSGi R4 Compendium Bundle (4)
-> start 4 5 6 7 8 9
-> ps
START LEVEL 1
   ID   State         Level  Name
[   0] [Active     ] [    0] System Bundle (1.8.0)
[   1] [Active     ] [    1] Apache Felix Shell Service (1.2.0)
[   2] [Active     ] [    1] Apache Felix Shell TUI (1.2.0)
[   3] [Active     ] [    1] Apache Felix Bundle Repository (1.4.0)
[   4] [Active     ] [    1] Servlet 2.1 API (1.0.0)
[   5] [Active     ] [    1] Apache Felix Log Service (1.0.0)
[   6] [Active     ] [    1] Apache Felix Declarative Services (1.0.8)
[   7] [Active     ] [    1] HTTP Service (1.0.1)
[   8] [Active     ] [    1] Apache Felix Web Management Console (1.2.10)
[   9] [Active     ] [    1] OSGi R4 Compendium Bundle (4)
->
```

Now, if you load http://localhost:8080/system/console in a web browser and supply the default Felix administrative credentials (username "admin" and password "admin"), you will see the Apache Felix administrative web interface. Now that you've made use of the Nexus Proxy repository for the remote Felix OBR repository, the next time you browse the contents of the Felix OBR proxy repository, you are going to see that Nexus has cached all of the referenced bundles as shown in Figure 1.9, “Cached Bundles from a Remote OBR Repository”.

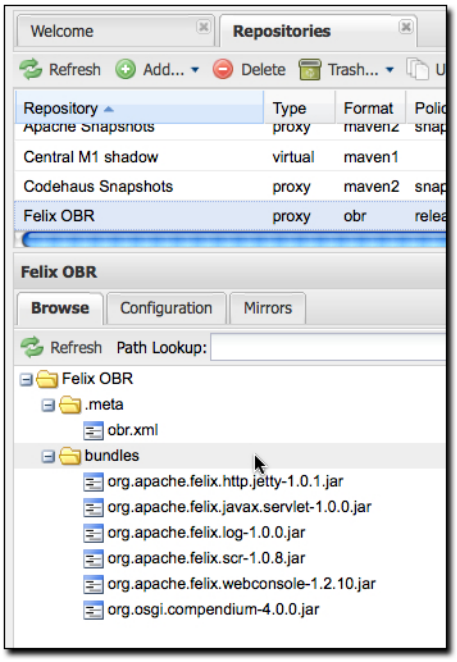

Figure 1.9. Cached Bundles from a Remote OBR Repository

1.11. Grouping OSGi Bundle Repositories

So far in this chapter, you've deployed an artifact to a Maven repository and consumed it using a Virtual OBR repository as a bridge between Maven and OSGi. In the previous recipe, you proxied a remote OBR repository using a Nexus proxy repository. In this recipe, you are going to learn how to consolidate multiple OBR repositories into a single URL using a Nexus repository group.

1.11.1. Task

You need to consolidate a number of Nexus OBR repositories into a single OBR repository interface. You have a collection of Proxy, Virtual, and Hosted OBR repositories, and you want to connect your OSGi aware systems to a single OBR repository, reducing the number of moving parts involved in your current deployment procedures.

1.11.2. Action

Create a OBR repository group in Nexus Professional which combines one or more OBR repositories into a single repository. To do this:

1. Load the Nexus interface in a web browser by opening the URL http://localhost:8081/nexus.
2. Login as an administrative user using the default credentials of admin/admin123 (if you haven't already changed the default password).
3. Click on Repositories in the left navigation menu.
4. Click on the Add.. button above the list of Nexus repositories and groups.
5. Select "Repository Group" from the resulting dropdown.
6. In the New Repository Group window, supply the following values as shown in Figure 1.10, “Creating a New OBR Group”:
   a. Group ID: **`obr-group`**
   b. Group Name: **`OBR Group`**

c. Provider: **`OBR Group`**

7. Drag the two OBR repositories created in the previous recipes to the Ordered Group Repositories list.

8. Click the Save button to create the new Repository Group.

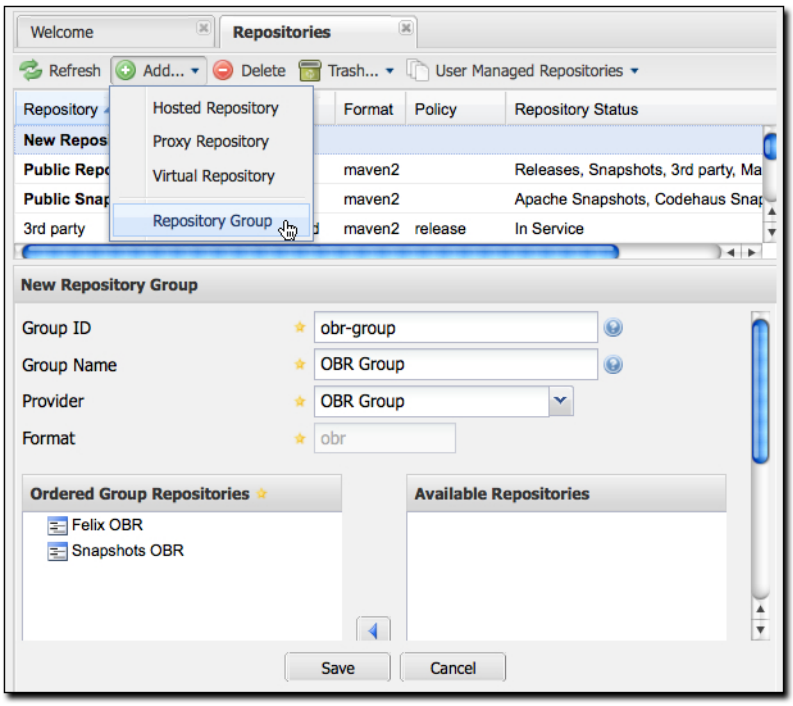

Figure 1.10. Creating a New OBR Group

1.11.3. Detail

To verify that the new OBR repository group is generating the the appropriate XML combining the Felix OBR reposiotry and the Snapshots OBR repository, click on Repositories in the left navigation menu, and then select "OBR Group" from the list of repositories. If you look at the Browse tab for this repository and drill into the `.meta/` directory, you can then right-click on the `obr.xml` file and select Download from the context menu. The URL for this group's repository XML should be http://localhost:8081/nexus/content/groups/obr-group/.meta/obr.xml. Again,bBe warned, this is going to be an even larger XML file than the repository XML for the Felix OBR proxy. Search for mcookbook and note that the repository also contains references to all of the artifacts from the remote Felix OBR repository. The Nexus Professional OBR repository group has effectively merged the contents of these two repositories into a single unified interface which will simplify your system's interaction to a single URL.

If you want to see the contents of this repository group, install Felix and add a reference to the Nexus repository group:

1. Download Apache Felix from http://www.apache.org/dist/felix/felix-1.8.0.tar.gz.

2. Unpack the Felix distribution on your local workstation.

3. Change directory to the Felix directory.

4. Start Apache Felix with **`java -jar ./bin/felix.jar`**

5. Add the Snapshots repository as an OBR repository by copying the URL to the repository.xml and passing it to the obr command: **`obr add-url http://localhost:8081/nexus/content/groups/obr-group/.meta/obr.xml`**

6. List the contents of the OBR repository.

7. Verify that the group contains bundles from Felix and the bundle from the osgi-project.

These commands are captured in the following screen listing:

```
~/programs $ wget http://www.apache.org/dist/felix/felix-1.8.0.tar.gz
--2009-07-14 22:46:19--  http://www.apache.org/dist/felix/felix-1.8.0.tar.gz
Resolving www.apache.org... 140.211.11.130
Connecting to www.apache.org|140.211.11.130|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 901620 (880K) [application/x-gzip]
Saving to: `felix-1.8.0.tar.gz'
2009-07-14 22:46:20 (924 KB/s) - `felix-1.8.0.tar.gz' saved [901620/901620]

~/programs $ tar xvzf felix-1.8.0.tar.gz
felix-1.8.0/
felix-1.8.0/bin/
felix-1.8.0/bin/felix.jar
...
felix-1.8.0/NOTICE
~/programs $ cd felix-1.8.0
~/programs/felix-1.8.0 $ java -jar ./bin/felix.jar

Welcome to Felix.
=================

-> obr add-url http://localhost:8081/nexus/content/groups/obr-group/.meta/obr.xml
-> obr list
Apache Felix Bundle Repository (1.2.1, ...)
Apache Felix Configuration Admin Service (1.0.4, ...)
Apache Felix Declarative Services (1.0.8, ...)
Apache Felix EventAdmin (1.0.0)
Apache Felix File Install (1.2.0, ...)
Apache Felix HTTP Service Jetty (1.0.1, ...)
Apache Felix iPOJO (1.2.0, ...)
Apache Felix iPOJO (0.8.0)
Apache Felix iPOJO Arch Command (1.2.0, ...)
Apache Felix iPOJO Composite (1.2.0)
Apache Felix iPOJO Composite (1.0.0, ...)
Apache Felix iPOJO Event Admin Handler (1.2.0, ...)
Apache Felix iPOJO Extender Pattern Handler (1.2.0)
Apache Felix iPOJO Extender Pattern Handler (1.0.0, ...)
Apache Felix iPOJO JMX Handler (1.2.0, ...)
Apache Felix iPOJO Temporal Service Dependency Handler (1.2.0, ...)
Apache Felix iPOJO White Board Pattern Handler (1.2.0, ...)
Apache Felix Log Service (1.0.0)
Apache Felix Metatype Service (1.0.2, ...)
Apache Felix Prefrences Service (1.0.2)
Apache Felix Remote Shell (1.0.2)
Apache Felix Shell Service (1.0.2, ...)
Apache Felix Shell TUI (1.0.2, ...)
Apache Felix UPnP Base Driver (0.8.0)
Apache Felix UPnP Extra (0.4.0)
Apache Felix UPnP Tester (0.4.0)
Apache Felix Web Management Console (1.2.10, ...)
org.sonatype.mcookbook (1.0.0.SNAPSHOT, ...)
OSGi OBR Service API (1.0.0)
OSGi R4 Compendium Bundle (4.0.0)
Servlet 2.1 API (1.0.0)
->
```

As anticipated, the list of bundles in the group repository contains all of the bundles from the Apache Felix repository and the bundle from the Snapshots OBR repository - `org.sonatype.mcookbook`.

Chapter 2. Groovy Maven

2.1. Introduction

Groovy is just that: Groovy, and maybe in future versions of this book, I'll go on and on and on about how wonderful it is. For now, the pre-alpha Maven Cookbook gets straight to the content. This chapter contains a number of recipes that should make it easier for you to integrate Groovy into your Maven builds. For more information about Groovy, take a look at the Groovy project page http://groovy.codehaus.org/.

2.2. Running an Inline Groovy Script in a Maven Build

While Maven covers a lot of ground, there are certainly times when you just want to shell out to a script to get something simple done without having to write a custom plugin. Groovy is ideal for these situations because it provides you with a simple, dynamic scripting language that can be added directly to a project's POM.

2.2.1. Task

You need to run some Groovy script as a part of your build process.

2.2.2. Action

Put some Groovy in your project's POM like this:

Example 2.1. Running a Groovy Script from a POM

```
<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/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.sonatype.mcookbook</groupId>
  <artifactId>groovy-script</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>groovy-script</name>
  <dependencies>
    <dependency>
      <groupId>org.apache.maven</groupId>
      <artifactId>maven-model</artifactId>
      <version>2.2.0</version>
    </dependency>
  </dependencies>
  <build>
    <plugins>
      <plugin>
        <groupId>org.codehaus.groovy.maven</groupId>
        <artifactId>gmaven-plugin</artifactId>
        <executions>
          <execution>
            <id>groovy-magic</id>
            <phase>prepare-package</phase>
            <goals>
              <goal>execute</goal>
            </goals>
            <configuration>
              <source>
                def depFile = new File(project.build.outputDirectory, 'deps.txt')
```

```
                project.dependencies.each() {
                  depFile.write("${it.groupId}:${it.artifactId}:${it.version}")
                }

                ant.copy(todir: project.build.outputDirectory ) {
                  fileset(dir: project.build.sourceDirectory)
                }
              </source>
            </configuration>
          </execution>
        </executions>
      </plugin>
    </plugins>
  </build>
</project>
```

Example 2.1, “Running a Groovy Script from a POM” configures the GMaven Plugin's execute goal to execute during the prepare-package phase. The source configuration supplies a Groovy script which is run during the execution of the execute goal. Also note that the execution has an id element with a value of groovy-magic. This id element isn't required if you are only configuring one execution for a plugin, but it is necessary once you define more than one execution for a given plugin.

The Groovy script included, creates a file named "deps.txt" in `${basedir}/target/classes`, it then iterates through the project's declared dependencies, and then it copies all of the source in `${basedir}/src/main/java` into the `${basedir}/target/classes` directory. This script demonstrates the use of Groovy's closure syntax and ease with which you can manipulate the filesystem. Groovy is an ideal scripting language for manipulating text and working with files. To test this script example, run **`mvn package`** as shown below.

```
~/examples/groovy/groovy-script $ mvn package -Dmaven.test.skip=true
[INFO] Scanning for projects...
[INFO] ------------------------------------------------------------------------
[INFO] Building groovy-script
[INFO]    task-segment: [package]
[INFO] ------------------------------------------------------------------------
[INFO] [resources:resources]
[WARNING] Using platform encoding (MacRoman actually) to copy filtered resources, i.e. build is platform
          dependent!
[INFO] skip non existing resourceDirectory ~/examples/groovy/groovy-script/src/main/resources
[INFO] [compiler:compile]
[INFO] Nothing to compile - all classes are up to date
[INFO] [resources:testResources]
[WARNING] Using platform encoding (MacRoman actually) to copy filtered resources, i.e. build is platform
          dependent!
[INFO] skip non existing resourceDirectory ~/examples/groovy/groovy-script/src/test/resources
[INFO] [compiler:testCompile]
[INFO] Not compiling test sources
[INFO] [surefire:test]
[INFO] Tests are skipped.
[INFO] [groovy:execute {execution: default}]
[INFO] [jar:jar]
[INFO] Building jar: ~/examples/groovy/groovy-script/target/groovy-script-1.0-SNAPSHOT.jar
```

After running this example, list the contents of `${basedir}/target/classes/deps.txt`, and you should see the following contents:

```
~/examples/groovy/groovy-script $ more deps.txt
org.apache.maven:maven-model:2.2.0
```

2.2.3. Detail

This simple Groovy script contained a number of references to implicit objects such as project and ant, it also demonstrated the simple syntax of Groovy and its closure-friendly nature of Groovy.

2.3. Executing Groovy Scripts in a Maven Build

2.3.1. Task

You need to execute one or more groovy scripts in a Maven build.

2.3.2. Action

Configure the execute goal of the GMaven plugin, reference the Groovy script in the source configuration for the execution. The example POM shown in Example 2.2, “Executing External Groovy Scripts in a Maven Build” configures two executions of the execute goal referencing two scripts stored in `${basedir}/src/main/groovy`.

Example 2.2. Executing External Groovy Scripts in a Maven Build

```
<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/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.sonatype.mcookbook</groupId>
  <artifactId>groovy-script-ex</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>groovy-script-ex</name>
  <dependencies>
    <dependency>
      <groupId>org.apache.maven</groupId>
      <artifactId>maven-model</artifactId>
      <version>2.2.0</version>
    </dependency>
  </dependencies>
  <build>
    <plugins>
      <plugin>
        <groupId>org.codehaus.groovy.maven</groupId>
        <artifactId>gmaven-plugin</artifactId>
        <executions>
          <execution>
            <id>create-deps-file</id>
            <phase>process-classes</phase>
            <goals>
              <goal>execute</goal>
            </goals>
            <configuration>
              <source>${basedir}/src/main/groovy/CreateDeps.groovy</source>
            </configuration>
          </execution>
          <execution>
            <id>copy-the-source</id>
            <phase>prepare-package</phase>
            <goals>
              <goal>execute</goal>
            </goals>
            <configuration>
              <source>${basedir}/src/main/groovy/CopySource.groovy</source>
            </configuration>
          </execution>
        </executions>
      </plugin>
    </plugins>
  </build>
</project>
```

The `CreateDeps.groovy` script creates a file named `deps.txt` in `${basedir}/target/classes` which contains a list of direct project dependencies, and the `CopySource.groovy` script copies the source from `${basedir}/src/main/java` to `${basedir}/target/classes`.

Example 2.3. The CreateDeps.groovy Script

```
def depFile = new File(project.build.outputDirectory,
              'deps.txt')

project.dependencies.each() {
  depFile.write("${it.groupId}:${it.artifactId}:${it.version}")
}
```

Example 2.4. The CopySource.groovy Script

```
ant.copy(todir: project.build.outputDirectory ) {
  fileset(dir: project.build.sourceDirectory)
}
```

2.4. Writing Plugins in Groovy

Groovy is a dynamic language based on the Java Virtual Machine which compiles to Java bytecode. Groovy is a project in the Codehaus community. If you are fluent in Java, Groovy will seem like a natural choice for a scripting language. Groovy takes the features of Java, pares down the syntax a bit, and adds features like closures, duck-typing, and regular expressions. For more information about Groovy, please see the Groovy web site at http://groovy.codehaus.org.

2.4.1. Creating a Groovy Plugin

To create a Maven Plugin using Groovy, you only need two files: a `pom.xml` and a single Mojo implemented in Groovy. To get started, create a project directory named `firstgroovy-maven-plugin`. Place the following `pom.xml` in this directory.

Example 2.5. POM for a Groovy Maven Plugin

```
<?xml version="1.0" encoding="UTF-8"?>
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.sonatype.mavenbook.plugins</groupId>
  <artifactId>firstgroovy-maven-plugin</artifactId>
  <name>Example Groovy Mojo - firstgroovy-maven-plugin</name>
  <packaging>maven-plugin</packaging>
  <version>1.0-SNAPSHOT</version>
  <dependencies>
    <dependency>
      <groupId>org.codehaus.mojo.groovy</groupId>
      <artifactId>groovy-mojo-support</artifactId>
      <version>1.0-beta-3</version>
    </dependency>
  </dependencies>
  <build>
    <plugins>
      <plugin>
        <artifactId>maven-plugin-plugin</artifactId>
        <version>2.4</version>
      </plugin>
      <plugin>
        <groupId>org.codehaus.mojo.groovy</groupId>
        <artifactId>groovy-maven-plugin</artifactId>
        <version>1.0-beta-3</version>
        <extensions>true</extensions>
        <executions>
          <execution>
            <goals>
```

```
            <goal>generateStubs</goal>
            <goal>compile</goal>
            <goal>generateTestStubs</goal>
            <goal>testCompile</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
  </plugins>
 </build>
</project>
```

What's going on in this POM? First, notice that the packaging of the POM is `maven-plugin` because we are creating a project that will package a Maven plugin. Next, note that the project depends on the `groovy-mojo-support` artifact in the `org.codehaus.mojo.groovy` group.

Then under `src/main/groovy` in a directory `org/sonatype/mavenbook/plugins`, create a file named `EchoMojo.groovy` which contains the EchoMojo class.

Example 2.6.

```
package org.sonatype.mavenbook.plugins

import org.codehaus.mojo.groovy.GroovyMojo

/**
 * Example goal which echos a message
 *
 * @goal echo
 */
class EchoMojo extends GroovyMojo {

    /**
     * Message to print
     *
     * @parameter expression="${echo.message}"
     *            default-value="Hello Maven World"
     */
    String message

    void execute() {
      log.info( message )
    }
}
```

Chapter 3. Scala and Maven

3.1. Introduction

Scala., haven't used it yet? Not surprised, few have, but everyone seems to be all atwitter about how amazing and transformative it is, so maybe you should take a look? If you are interested in Scala and you already use Maven, you are in luck. The Scala community has spent a great deal of effort making sure that the Maven integration is first-class. This chapter will introduce you to Scala by way of Maven.

3.2. Running an Inline Scala Script in a Maven Build

3.2.1. Task

You need to execute an inline Scala script as a part of your Maven build.

3.2.2. Action

Configure the `script` goal of the `maven-scala-plugin`, and pass a Scala script to the `script` configuration parameter.

Example 3.1. Executing an Inline Scala Script with the Maven Scala Plugin

```
<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/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.sonatype.mcookbook</groupId>
  <artifactId>scala-script</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>scala-script</name>
  <dependencies>
    <dependency>
      <groupId>org.apache.maven</groupId>
      <artifactId>maven-model</artifactId>
      <version>2.2.0</version>
    </dependency>
    <dependency>
      <groupId>org.scala-lang</groupId>
      <artifactId>scala-library</artifactId>
      <version>2.7.3</version>
    </dependency>
    <dependency>
      <groupId>org.scala-tools</groupId>
      <artifactId>maven-scala-plugin</artifactId>
      <version>2.10.1</version>
    </dependency>
    <dependency>
      <groupId>org.scalaforge</groupId>
      <artifactId>scalax</artifactId>
      <version>0.1</version>
    </dependency>
  </dependencies>
  <build>
    <plugins>
      <plugin>
        <groupId>org.scala-tools</groupId>
        <artifactId>maven-scala-plugin</artifactId>
        <version>2.10.1</version>
        <executions>
          <execution>
```

```
            <phase>prepare-package</phase>
            <goals>
              <goal>script</goal>
            </goals>
            <configuration>
              <keepGeneratedScript>true</keepGeneratedScript>
              <script>
                <![CDATA[
        import java.io.{File,  PrintWriter, FileWriter};
        import scalax.io.FileExtras;
        import scala.collection.mutable.HashSet;

        val outputDir = project.getBuild().getOutputDirectory();
        val depsFile = new FileExtras( new File( outputDir, "deps.txt" ) )
        val pw = depsFile.printWriter
        val depSet = new HashSet[String]
        for( d <- project.getDependencies() ) {
          depSet += d.getGroupId + ":" + d.getArtifactId + ":" + d.getVersion
        }
        pw.writeLines( depSet.toSeq )

                ]]>
              </script>
            </configuration>
          </execution>
        </executions>
      </plugin>
    </plugins>
  </build>
  <repositories>
    <repository>
      <id>scala-tools</id>
      <url>http://scala-tools.org/repo-releases/</url>
    </repository>
  </repositories>
</project>
```

If you run **`mvn package`**, the `script` goal will be executed during the `prepare-package` phase.

```
~/examples/scala/scala-script $ mvn package -Dmaven.test.skip=true
[INFO] Scanning for projects...
[INFO] ------------------------------------------------------------------------
[INFO] Building scala-script
[INFO]    task-segment: [package]
[INFO] ------------------------------------------------------------------------
...
[INFO] [scala:script {execution: scala-magic}]
[INFO] Checking for multiple versions of scala
...
[INFO] BUILD SUCCESSFUL
```

After this script executes, there will be a file named `deps.txt` in `${basedir}/target/classes` containing a list of the project's direct dependencies.

```
org.scalaforge:scalax:0.1
org.apache.maven:maven-model:2.2.0
org.scala-tools:maven-scala-plugin:2.10.1
org.scala-lang:scala-library:2.7.3
```

3.2.3. Detail

Example 3.1, “Executing an Inline Scala Script with the Maven Scala Plugin” includes the scala-tools repository and the dependency on version 0.1 of scalax to gain access to the FileExtras class which is used to augment the methods available on a `File` object. The FileHelper wraps the File object, and adds the method `writeLines()` which is used to write a Sequence of String objects. This Sequence of String objects is stored in a HashSet which is populated in a for loop that loops over all of the project's dependencies.

The other piece of configuration in this example is the keepGeneratedScript option. This option tells the Maven Scala plugin to retain the temporary Scala source which is generated by the plugin. The Scala plugin takes the inline Scala source code and creates a temporary script in ${basedir}/target/.scalaScriptGen/embeddedScript_1.scala. If keepGeneratedScript is set to false, the Maven Scala plugin will delete this file when it is finished executing the script. If keepGeneratedScript is set to true, the Scala plugin will retain this source file. Let's take a look at the embeddedScript_1.scala file:

```
import scala.collection.jcl.Conversions._
class embeddedScript_1(project:org.scala_tools.maven.model.MavenProjectAdapter,log:org
.apache.maven.plugin.logging.Log) {
  import java.io.{File,  PrintWriter, FileWriter};
          import scalax.io.FileExtras;
          import scala.collection.mutable.HashSet;

          val outputDir = project.getBuild().getOutputDirectory();
          val depsFile = new FileExtras( new File( outputDir, "deps.txt" ) )
          val pw = depsFile.printWriter
        val depSet = new HashSet[String]
          for( d <- project.getDependencies() ) {
            depSet += d.getGroupId + ":" + d.getArtifactId + ":" + d.getVersion
          }
          pw.writeLines( depSet.toSeq )
}
```

Notice that the class includes to argument project:org.scala_tools.maven.model.MavenProjectAdapter and log:org.apache.maven.plugin.logging.Log. The project argument gives you access to the Maven project object model and the log argument gives you access to the logging facilities of Maven. The project and log objects are only available when the Maven Scala plugin is available on the classpath, this is why you see the Maven Scala plugin declared both as a dependency of the project under dependencies in Example 3.1, “Executing an Inline Scala Script with the Maven Scala Plugin” and as a plugin.

3.3. Running an External Scala Script in a Maven Build

3.3.1. Task

You need to execute an external Scala script as a part of your Maven build.

3.3.2. Action

Similar to Example 3.1, “Executing an Inline Scala Script with the Maven Scala Plugin”, configure the `script` goal of the Maven Scala plugin and specify a Scala script in the `scriptFile` configuration parameter.

Example 3.2. Configuring the Maven Scala Plugn Script Goal to Execute an External Script

```
<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/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.sonatype.mcookbook</groupId>
  <artifactId>scala-script-ex</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>scala-script-ex</name>
  <dependencies>
    <dependency>
      <groupId>org.apache.maven</groupId>
      <artifactId>maven-model</artifactId>
      <version>2.2.0</version>
    </dependency>
    <dependency>
      <groupId>org.scala-lang</groupId>
      <artifactId>scala-library</artifactId>
      <version>2.7.3</version>
```

```
    </dependency>
    <dependency>
      <groupId>org.scala-tools</groupId>
      <artifactId>maven-scala-plugin</artifactId>
      <version>2.10.1</version>
    </dependency>
    <dependency>
      <groupId>org.scalaforge</groupId>
      <artifactId>scalax</artifactId>
      <version>0.1</version>
    </dependency>
  </dependencies>
  <build>
    <plugins>
      <plugin>
        <groupId>org.scala-tools</groupId>
        <artifactId>maven-scala-plugin</artifactId>
        <version>2.10.1</version>
        <executions>
          <execution>
            <id>scala-magic</id>
            <phase>prepare-package</phase>
            <goals>
              <goal>script</goal>
            </goals>
            <configuration>
              <keepGeneratedScript>true</keepGeneratedScript>
              <scriptFile>${basedir}/src/main/scala/CreateDeps.scala</scriptFile>
            </configuration>
          </execution>
        </executions>
      </plugin>
    </plugins>
  </build>
  <repositories>
    <repository>
      <id>scala-tools</id>
      <url>http://scala-tools.org/repo-releases/</url>
    </repository>
  </repositories>
</project>
```

The script that this build will execute is the same Scala script that was executed in Example 3.1, “Executing an Inline Scala Script with the Maven Scala Plugin”. In this example, the script is stored in an external file in `${basedir}/src/main/scala`, and in Example 3.1, “Executing an Inline Scala Script with the Maven Scala Plugin”, the script was listed inline in the project's POM.

Example 3.3. The CreateDeps.scala Script

```
import java.io.{File,  PrintWriter, FileWriter};
import scalax.io.FileExtras;
import scala.collection.mutable.HashSet;

val outputDir = project.getBuild().getOutputDirectory();
val depsFile = new FileExtras( new File( outputDir, "deps.txt" ) );
val pw = depsFile.printWriter;
val depSet = new HashSet[String];
for( d <- project.getDependencies() ) {
  depSet += d.getGroupId + ":" + d.getArtifactId + ":" + d.getVersion;
}
pw.writeLines( depSet.toSeq );
```

Just like the example in Section 3.2, “Running an Inline Scala Script in a Maven Build”, this script creates a file named `deps.txt` in `${basedir}/target/classes` which contains a list of the project's dependencies.

```
org.scalaforge:scalax:0.1
org.apache.maven:maven-model:2.2.0
```

```
org.scala-tools:maven-scala-plugin:2.10.1
org.scala-lang:scala-library:2.7.3
```

3.3.3. Detail

There is very little difference between the example from Section 3.2, “Running an Inline Scala Script in a Maven Build” and the example in this section other than the location of the Scala script. If you look in `${basedir}/target/.scalaScriptGen` you can see that the Scala plugin creates a file named `CreateDeps_1.scala` that contains a class named `CreateDeps_1`.

Chapter 4. Ant and Maven

4.1. Introduction

Ant and Maven? Yes, it is true.

Ant isn't a language as much as it is a build tool which allows you to describe a build as a set of tasks grouped into build targets. Ant then allows you to declare dependencies between build targets. When you are using Ant you are essentially creating your own lifecycle. An Ant `build.xml` might have an install target which depends on a test target which depends on a compile target, but these relationships are arbitrary and managed by the person who writes the Ant build file.

Ant is something of a ancestor to Maven, it was the ubiquitous procedural build tool that almost every project used before Maven introduced the concept of wide-scale reusability of common build plugins and the concept of a universal lifecycle.

4.2. Running an Inline Ant Script in a Maven Build

4.2.1. Task

You need to run an inline Ant script as a part of your Maven build.

4.2.2. Action

Configure the `run` goal of the Maven AntRun plugin to execute an inline Ant script. The POM shown in configures the `run` goal to execute during the `prepare-package` phase of the build. The Ant script is contained in the `tasks` configuration parameter for the `ant-magic` execution.

Example 4.1. Executing an Inline Ant Script in a Maven Build

```
<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/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.sonatype.mcookbook</groupId>
  <artifactId>ant-script</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>ant-script</name>
  <build>
    <plugins>
      <plugin>
        <artifactId>maven-antrun-plugin</artifactId>
        <executions>
          <execution>
            <id>ant-magic</id>
            <phase>prepare-package</phase>
            <goals>
              <goal>run</goal>
            </goals>
            <configuration>
              <tasks>
                <property name="compile_classpath"
                          refid="maven.compile.classpath"/>
                <echo file="${project.build.outputDirectory}/deps.txt"
                      message="compile classpath: ${compile_classpath}"/>

                <copy todir="${project.build.outputDirectory}">
                  <fileset dir="${project.build.sourceDirectory}"/>
                </copy>
              </tasks>
```

```
            </configuration>
          </execution>
        </executions>
      </plugin>
    </plugins>
  </build>
</project>
```

You can run this script, by executing the package phase.

```
~/examples/ant/ant-script $ mvn package
[INFO] Scanning for projects...
...
[INFO] [antrun:run {execution: ant-magic}]
[INFO] Executing tasks
     [copy] Copying 1 file to ~/examples/ant/ant-script/target/classes
[INFO] Executed tasks
...
[INFO] BUILD SUCCESSFUL
```

Once this simple Ant script has been completed, the contents of `${basedir}/target/classes/deps.txt` should contain the following:

```
compile classpath: ~/examples/ant/ant-script/target/classes
```

4.3. Running an External Ant Script in a Maven Build

4.3.1. Task

You need to execute an external Ant script in a Maven build.

4.3.2. Action

Configure the run goal form the Maven AntRun plugin. Define any properties you wish to pass to the external Ant build, and then call the external Ant build with the ant task, specifying the antfile and target you wish to execute.

Example 4.2. Executing an External Ant Script from a Maven Build

```
<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/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.sonatype.mcookbook</groupId>
  <artifactId>ant-script-ex</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>ant-script-ex</name>
  <build>
    <plugins>
      <plugin>
        <artifactId>maven-antrun-plugin</artifactId>
        <executions>
          <execution>
            <id>ant-magic</id>
            <phase>prepare-package</phase>
            <goals>
              <goal>run</goal>
            </goals>
            <configuration>
              <tasks>
                <property name="compile_classpath"
                          refid="maven.compile.classpath"/>
```

```
                <property name="outputDir"
                          value="${project.build.outputDirectory}"/>
                <property name="sourceDir"
                          value="${project.build.sourceDirectory}"/>
                <ant antfile="${basedir}/src/main/ant/create-deps.xml"
                     target="create"/>
              </tasks>
            </configuration>
          </execution>
        </executions>
      </plugin>
    </plugins>
  </build>
</project>
```

To execute this external Ant build, run **`mvn package`**.

```
~/examples/ant/ant-script Tim$ mvn package
[INFO] Scanning for projects...
[INFO] ------------------------------------------------------------------------
[INFO] Building ant-script-ex
...
[INFO] [antrun:run {execution: ant-magic}]
[INFO] Executing tasks

create:
     [copy] Copying 1 file to ~/examples/ant/ant-script-ex/target/classes
[INFO] Executed tasks
...
[INFO] BUILD SUCCESSFUL
```

Once this simple Ant script has been completed, the contents of `${basedir}/target/classes/deps.txt` should contain the following:

```
compile classpath: ~/examples/ant/ant-script-ex/target/classes
```

4.4. Creating an Ant Maven Plugin

While Maven is an improvement on Ant, Ant can still be useful when describing parts of the build process. Ant provides a set of tasks which can come in handy when you need to perform file operations or XSLT transformations or any other operation you could think of. There is a large library of available Ant tasks for everything from running JUnit tests to transforming XML to copying files to a remote server using SCP. An overview of available Ant tasks can be found online in the Apache Ant Manual[1]. You can use these tasks as a low-level build customization language, and you can also write a Maven plugin where, instead of a Mojo written in Java, you can pass parameters to a Mojo which is an Ant build target.

4.4.1. Task

You need to write a Maven Plugin in Ant.

4.4.2. Action

To create a Maven plugin using Ant, you will need to have a `pom.xml` and a single Mojo implemented in Ant. To get started, create a project directory named firstant-maven-plugin. Place the following `pom.xml` in this directory.

Example 4.3. POM for an Ant Maven Plugin

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.sonatype.mavenbook.plugins</groupId>
```

[1] http://ant.apache.org/manual/tasksoverview.html

```
  <artifactId>firstant-maven-plugin</artifactId>
  <name>Example Ant Mojo - firstant-maven-plugin</name>
  <packaging>maven-plugin</packaging>
  <version>1.0-SNAPSHOT</version>
  <dependencies>
    <dependency>
      <groupId>org.apache.maven</groupId>
      <artifactId>maven-script-ant</artifactId>
      <version>${maven.version}</version>
    </dependency>
  </dependencies>
  <build>
    <plugins>
      <plugin>
        <artifactId>maven-plugin-plugin</artifactId>
        <version>2.4</version>
        <dependencies>
          <dependency>
            <groupId>org.apache.maven.plugin-tools</groupId>
            <artifactId>maven-plugin-tools-ant</artifactId>
            <version>2.4</version>
          </dependency>
        </dependencies>
      </plugin>
    </plugins>
  </build>
</project>
```

Next, you will need to create your Ant Mojo. An Ant mojo consists of two parts: the Ant tasks in an XML file, and a file which supplies Mojo descriptor information. The Ant plugin tools are going to look for both of these files in `${basedir}/src/main/scripts`. One file will be named `echo.build.xml` and it will contain the Ant XML.

Example 4.4. Echo Ant Mojo

```
<project>
  <target name="echotarget">
    <echo>${message}</echo>
  </target>
</project>
```

The other file will describe the Echo Ant Mojo and will be in the `echo.mojos.xml` file also in `${basedir}/src/main/scripts`.

Example 4.5. Echo Ant Mojo Descriptor

```
<pluginMetadata>
  <mojos>
    <mojo>
      <goal>echo</goal>
      <call>echotarget</call>
      <description>Echos a Message</description>
      <parameters>
        <parameter>
          <name>message</name>
          <property>message</property>
          <required>false</required>
          <expression>${message}</expression>
          <type>java.lang.Object</type>
          <defaultValue>Hello Maven World</defaultValue>
          <description>Prints a message</description>
        </parameter>
      </parameters>
    </mojo>
  </mojos>
</pluginMetadata>
```

This `echo.mojos.xml` file configures the Mojo descriptor for this plugin. It supplies the goal name "echo", and it tells Maven what Ant task to call in the call element. In addition to configuring the description, this XML file configures the message parameter to use the expression `${message}` and to have a default value of "Hello Maven World."

If you've configured your plugin groups in `~/.m2/settings.xml` to include `org.sonatype.mavenbook.plugins`, you can install this Ant plugin by executing the following command at the command-line:

```
$ mvn install
[INFO] ------------------------------------------------------------------------
[INFO] Building Example Ant Mojo - firstant-maven-plugin
[INFO]    task-segment: [install]
[INFO] ------------------------------------------------------------------------
[INFO] [plugin:descriptor]
[INFO] Using 3 extractors.
[INFO] Applying extractor for language: java
[INFO] Extractor for language: java found 0 mojo descriptors.
[INFO] Applying extractor for language: bsh
[INFO] Extractor for language: bsh found 0 mojo descriptors.
[INFO] Applying extractor for language: ant
[INFO] Extractor for language: ant found 1 mojo descriptors.
...
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
```

Note that the `plugin:descriptor` goal found a single Ant mojo descriptor. To run this goal, you would execute the following command-line:

```
$ mvn firstant:echo
...
[INFO] [firstant:echo]

echotarget:
     [echo] Hello Maven World
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
```

The `echo` goal executed and printed out the default value of the `message` parameter. If you are used to Apache Ant build scripts, you will notice that Ant prints out the name of the target executed and then adds a logging prefix to the output of the echo Ant task.

4.4.3. Detail

While Maven is an improvement on Ant, Ant can still be useful when describing parts of the build process. Ant provides a set of tasks which can come in handy when you need to perform file operations or XSLT transformations or any other operation you could think of. There is a large library of available Ant tasks for everything from running JUnit tests to transforming XML to copying files to a remote server using SCP. An overview of available Ant tasks can be found online in the Apache Ant Manual[2]. You can use these tasks as a low-level build customization language, and you can also write a Maven plugin where, instead of a Mojo written in Java, you can pass parameters to a Mojo which is an Ant build target.

[2] http://ant.apache.org/manual/tasksoverview.html

Chapter 5. Ruby and Maven

5.1. Introduction

Ant and Maven? Yes, it is true.

5.2. Writing Plugins in JRuby

Ruby is an object-oriented scripting language which provides a rich set of facilities for meta-programming and reflection. Ruby's reliance on closures and blocks make for a programming style that is both compact and powerful. Although Ruby has been around since 1993, most people came to know Ruby after it was made popular by a Ruby-based web framework known as Ruby on Rails. JRuby is a Ruby interpreter written in Java. For more information about the Ruby language, see: http://www.ruby-lang.org/, and for more information about JRuby, see: http://jruby.codehaus.org/.

5.2.1. Creating a JRuby Plugin

To create a Maven plugin using JRuby, you will need to have a `pom.xml` and a single Mojo implemented in Ruby. To get started, create a project directory named `firstruby-maven-plugin`. Place the following `pom.xml` in this directory.

Example 5.1. POM for a JRuby Maven Plugin

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.sonatype.mavenbook.plugins</groupId>
  <artifactId>firstruby-maven-plugin</artifactId>
  <name>Example Ruby Mojo - firstruby-maven-plugin</name>
  <packaging>maven-plugin</packaging>
  <version>1.0-SNAPSHOT</version>
  <dependencies>
    <dependency>
      <groupId>org.codehaus.mojo</groupId>
      <artifactId>jruby-maven-plugin</artifactId>
      <version>1.0-beta-4</version>
      <scope>runtime</scope>
    </dependency>
  </dependencies>
  <build>
    <plugins>
      <plugin>
        <artifactId>maven-plugin-plugin</artifactId>
        <version>2.4</version>
        <dependencies>
          <dependency>
            <groupId>org.codehaus.mojo</groupId>
            <artifactId>jruby-maven-plugin</artifactId>
            <version>1.0-beta-4</version>
          </dependency>
        </dependencies>
      </plugin>
    </plugins>
  </build>
</project>
```

Next, you will need to create a Mojo implemented in Ruby. Maven is going to look for a Ruby Mojo in `${basedir}/src/main/scripts`. Put the following Ruby class in `${basedir}/src/main/scripts/echo.rb`.

Example 5.2. The Echo Ruby Mojo

```
# Prints a message
```

```
# @goal "echo"
# @phase "validate"
class Echo < Mojo

  # @parameter type="java.lang.String" default-value="Hello Maven World" \
    expression="${message}"
  def message
  end

  def execute
    info $message
  end

end

run_mojo Echo
```

The `Echo` class must extend `Mojo`, and it must override the `execute()` method. At the end of the `echo.rb` file, you will need to run the mojo with "`run_mojo Echo`". To install this plugin, run mvn install:

```
$ mvn install
[INFO] Scanning for projects...
[INFO] ------------------------------------------------------------------------
[INFO] Building Example Ruby Mojo - firstruby-maven-plugin
[INFO]    task-segment: [install]
[INFO] ------------------------------------------------------------------------
...
[INFO] [plugin:descriptor]
...
[INFO] Applying extractor for language: jruby
[INFO] Ruby Mojo File: /echo.rb
[INFO] Extractor for language: jruby found 1 mojo descriptors.
...
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
```

During the build, you should see that the Maven Plugin Plugin's descriptor goal applies the JRuby extractor to create a `plugin.xml` which captures the annotations in the `Echo` class. If you've configured your default plugin groups to include `org.sonatype.mavenbook.plugins`, you should be able to run this echo goal with the following command-line:

```
$ mvn firstruby:echo
...
[INFO] [firstruby:echo]
[INFO] Hello Maven World
...
```

5.2.2. Ruby Mojo Implementations

Ruby Mojos are annotated using comments in Ruby source files. A single annotation like `@parameter` takes a number of attributes, and each of these attributes must be specified on the same line. There can be no line-breaks between an annotations attribute in the Ruby source. Both classes and parameters are annotated. Parameters are annotated with four annotations: `@parameter`, `@required`, `@readonly`, and `@deprecated`. The `@parameter` attribute takes the following attributes:

alias
: An alias for the parameter. An alternate name which can be used to populate the same parameter.

default-value
: Provides a default value to the parameter if the supplied value or the parameter expression produces a null result. In `echo.rb`, we specify the default as "Hello Maven World".

expression
: Contains an expression which can resolve to a Maven property or a System property.

type
: The fully qualified Java type of the parameter. If the type is not specified it will default to `java.lang.String`.

In addition to the `@parameter` annotation, a parameter can take the following annotations:

@required "<true|false>"
: Marks the parameter as being required. The default value is false.

@readonly "<true|false>"
: Marks the parameter as read-only. If this is true, you may not override the default value or the value from the expression from the command line. The default value is false.

@deprecated "<true|false>"
: Marks the parameter as deprecated. The default value is false.

Putting this altogether, a fully annotated message parameter from `echo.rb` would look like the following code:

```
# @parameter type="java.lang.String" default-value="Hello Maven World" \
  expression="${message}"
# @readonly true
# @required false
# @deprecated false
def message
end
```

Ruby Mojo classes are annotated with the following attributes:

@goal
: Specifies the name of the goal.

@phase
: The default phase to bind this goal to.

@requiresDependencyResolution
: True if the Mojo requires that dependencies be resolved before execution.

@aggregator
: Marks this mojo as an aggregator.

@execute
: Provides the opportunity to execute a goal or lifecycle phase before executing this Mojo. The @execute annotation takes the following attributes:

  goal
  : Name of the goal to execute

  phase
  : Name of the lifecycle phase to execute

  lifecycle
  : Name of the lifecycle (if other than default)

For an example of an annotated Mojo class, consider the following code example:

```
# Completes some build task
# @goal custom-goal
# @phase install
# @requiresDependencyResolution false
# @execute phase=compile
class CustomMojo < Mojo
```

```
   ...
end
```

Mojo parameters can reference Java classes and Maven properties. The following example shows you how to get access to the Maven Project object from a Ruby Mojo.

Example 5.3. Referencing a Maven Project from a Ruby Mojo

```
# This is a mojo description
# @goal test
# @phase validate
class Test < Mojo
  # @parameter type="java.lang.String" default-value="nothing" alias="a_string"
  def prop
  end

  # @parameter type="org.apache.maven.project.MavenProject" \
    expression="${project}"
  # @required true
  def project
  end

  def execute
    info "The following String was passed to prop: '#{$prop}'"
    info "My project artifact is: #{$project.artifactId}"
  end
end

run_mojo Test
```

In the previous example, we can access properties on the `Project` class using standard Ruby syntax. If you put `test.rb` in `firstruby-maven-plugin`'s `src/main/scripts` directory, install the plugin, and then run it, you will see the following output:

```
$ mvn install
...
[INFO] [plugin:descriptor]
[INFO] Using 3 extractors.
[INFO] Applying extractor for language: java
...
[INFO] Applying extractor for language: jruby
[INFO] Ruby Mojo File: /echo.rb
[INFO] Ruby Mojo File: /test.rb
[INFO] Extractor for language: jruby found 2 mojo descriptors.
...
$ mvn firstruby:test
...
[INFO] [firstruby:test]
[INFO] The following String was passed to prop: 'nothing'
[INFO] My project artifact is: firstruby-maven-plugin
```

5.2.3. Logging from a Ruby Mojo

To log from a Ruby Mojo, call the `info()`, `debug()`, and `error()` methods with a message.

```
# Tests Logging
# @goal logtest
# @phase validate
class LogTest < Mojo

  def execute
    info "Prints an INFO message"
    error "Prints an ERROR message"
    debug "Prints to the Console"
  end
```

```
end

run_mojo LogTest
```

5.2.4. Raising a MojoError

If there is an unrecoverable error in a Ruby Mojo, you will need to raise a `MojoError`. Example 5.4, “Raising a MojoError from a Ruby Mojo” shows you how to raise a `MojoError`. This example mojo prints out a message and then raises a `MojoError`.

Example 5.4. Raising a MojoError from a Ruby Mojo

```
# Prints a Message
# @goal error
# @phase validate
class Error < Mojo

  # @parameter type="java.lang.String" default-value="Hello Maven World" \
    expression="${message}"
  # @required true
  # @readonly false
  # @deprecated false
  def message
  end

  def execute
    info $message
    raise MojoError.new( "This Mojo Raised a MojoError" )
  end

end

run_mojo Error
```

Running this Mojo, produces the following output:

```
$ mvn firstruby:error
...
INFO] [firstruby:error]
[INFO] Hello Maven World
[ERROR] This Mojo Raised a MojoError
```

5.2.5. Referencing Plexus Components from JRuby

A Ruby Mojo can depend on a Plexus component. To do this, you would use the `expression` attribute of the `@parameter` annotation to specify a role and a hint for Plexus. The following example Ruby Mojo, depends upon an Archiver component which Maven will retrieve from Plexus.

Example 5.5. Depending on a Plexus Component from a Ruby Mojo

```
# This mojo tests plexus integration
# @goal testplexus
# @phase validate
class TestPlexus < Mojo

  # @parameter type="org.codehaus.plexus.archiver.Archiver" \
expression="${component.org.codehaus.plexus.archiver.Archiver#zip}"
  def archiver
  end

  def execute
    info $archiver
  end
end
```

```
run_mojo TestPlexus
```

Please note that the attributes for an annotation in a Ruby Mojo cannot span multiple lines. If you were to run this goal, you would see Maven attempt to retrieve a component from Plexus with a role of `org.codehaus.plexus.arhiver.Archiver` and a hint of `zip`.

Chapter 6. Web Development

This is a test

6.1. Running a Web Application in a Servlet Container

6.1.1. Task

You need to run a web application in a servlet container.

6.1.2. Action

Configure your web application's Maven project to inlcude the Maven Jetty plugin as shown in the following POM:

```
<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/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.sonatype.mcookbook</groupId>
  <artifactId>sample-web</artifactId>
  <packaging>war</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>sample-web Maven Webapp</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>servlet-api</artifactId>
      <version>2.5</version>
    </dependency>
  </dependencies>
  <build>
    <finalName>sample-web</finalName>
    <plugins>
      <plugin>
        <groupId>org.mortbay.jetty</groupId>
        <artifactId>maven-jetty-plugin</artifactId>
        <version>6.1.22</version>
      </plugin>
    </plugins>
  </build>
</project>
```

To start the web application in Jetty, run the run goal from the Maven Jetty plugin by running **`mvn jetty:run`**.

```
$ mvn jetty:run
[INFO] Scanning for projects...
[INFO] ----------------------------------------------------------------------
[INFO] Building sample-web Maven Webapp
[INFO]    task-segment: [jetty:run]
[INFO] ----------------------------------------------------------------------
[INFO] Preparing jetty:run
[INFO] [resources:resources {execution: default-resources}]
[WARNING] Using platform encoding (MacRoman actually) to copy filtered
resources, i.e. build is platform dependent!
[INFO] Copying 0 resource
[INFO] [compiler:compile {execution: default-compile}]
```

```
[INFO] Nothing to compile - all classes are up to date
[INFO] [resources:testResources {execution: default-testResources}]
[WARNING] Using platform encoding (MacRoman actually) to copy filtered
resources, i.e. build is platform dependent!
[INFO] skip non existing resourceDirectory ~/maven-cookbook/
mcookbook-examples/web/sample-web/src/test/resources
[INFO] [compiler:testCompile {execution: default-testCompile}]
[INFO] No sources to compile
[INFO] [jetty:run {execution: default-cli}]
[INFO] Configuring Jetty for project: sample-web Maven Webapp
[INFO] Webapp source directory = ~/maven-cookbook/mcookbook-examples/web/
sample-web/src/main/webapp
[INFO] Reload Mechanic: automatic
[INFO] Classes = ~/maven-cookbook/mcookbook-examples/web/sample-web/
target/classes
2009-11-28 19:25:18.129:INFO::Logging to STDERR via org.mortbay.log.StdErrLog
[INFO] Context path = /sample-web
[INFO] Tmp directory =  determined at runtime
[INFO] Web defaults = org/mortbay/jetty/webapp/webdefault.xml
[INFO] Web overrides =  none
[INFO] web.xml file = ~/maven-cookbook/mcookbook-examples/web/sample-web/
src/main/webapp/WEB-INF/web.xml
[INFO] Webapp directory = ~/maven-cookbook/mcookbook-examples/web/sample-web
/src/main/webapp
[INFO] Starting jetty 6.1.22 ...
2009-11-28 19:25:18.231:INFO::jetty-6.1.22
2009-11-28 19:25:18.405:INFO::No Transaction manager found - if your webapp
requires one, please configure one.
2009-11-28 19:25:18.800:INFO::Started SelectChannelConnector@0.0.0.0:8080
[INFO] Started Jetty Server
```

At this point, you can use a web browser to navigate to http://localhost:8080/sample-web/ to interact with the web application.

6.1.3. Detail

Consider a simple web application with a single index.jsp page that contains a form, and a single Servlet that calculates a number from the Fibonacci sequence.

Example 6.1. Simple Form Accepting an Index to Pass to the Fibonacci Servlet

```
<html>
<body>
<h2>Fibonacci Page</h2>
<form action="fib" method="GET">
  <p>Fetch Fibonacci Sequence Index:
  <input type="text" name="index" size="5"/></p>
  <input type="submit" value="Calculate"/>
</form>
</body>
</html>
```

The following class is the Servlet which calculates the Fibonacci sequence. It takes a single parameter index and simply prints of the number at the specified position of the Fibonacci sequence.

Example 6.2. Fibonacci Servlet which Calculates a Number from the Fibonacci Sequence

```
package org.sonatype.mcookbook;

import java.io.IOException;
import java.util.Enumeration;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class FibonacciServlet extends HttpServlet {
```

```
  protected void doGet(HttpServletRequest req, HttpServletResponse resp)
      throws ServletException, IOException {
    int index = Integer.parseInt(req.getParameter("index"));
    resp.getWriter().write( fib(index) + "" );
    resp.getWriter().flush();
    resp.getWriter().close();
  }

  public long fib(int n) {
    if (n <= 1) return n;
    else return fib(n-1) + fib(n-2);
  }
}
```

The following web.xml configures the FibonacciServlet to respond to the request path `<context>/fib`.

Example 6.3. Web Application Descriptor for sample-web

```
<!DOCTYPE web-app PUBLIC
 "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
 "http://java.sun.com/dtd/web-app_2_3.dtd" >

<web-app>
  <display-name>Archetype Created Web Application</display-name>
  <servlet>
    <servlet-name>fibonacci</servlet-name>
    <servlet-class>org.sonatype.mcookbook.FibonacciServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>fibonacci</servlet-name>
    <url-pattern>/fib</url-pattern>
  </servlet-mapping>
</web-app>
```

After running mvn:jetty, you can load the initial form by going to http://localhost:8080/sample-web/index.jsp. Populating the form with an index and pressing calculate will load the Fibonacci servlet and print out the number at that position of the sequence.

6.2. Configuring a Servlet Container

6.3. Starting a WAR Dependency in a Servlet Container

6.3.1. Task

You need to configure Maven to download and start a web application from a repository manager.

6.3.2. Action

Use the Maven Dependency plugin to copy the web application's WAR artifact to your project. Then configure the Maven Jetty plugin to execute the web application using the plugin's dependencies element to configure the classpath for Jetty.

```
<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/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.sonatype.mcookbook</groupId>
  <artifactId>start-jackrabbit</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>start-jackrabbit</name>
  <build>
    <plugins>
      <plugin>
```

```
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-dependency-plugin</artifactId>
        <executions>
          <execution>
            <id>default-cli</id>❶
            <configuration>
              <artifactItems>
                <artifactItem>
                  <groupId>org.apache.jackrabbit</groupId>
                  <artifactId>jackrabbit-webapp</artifactId>
                  <version>1.6.0</version>
                  <type>war</type>
                  <overWrite>true</overWrite>
                  <destFileName>jackrabbit-webapp.war</destFileName>
                </artifactItem>
              </artifactItems>
              <outputDirectory>
                ${project.build.directory}/war
              </outputDirectory>
              <overWriteReleases>true</overWriteReleases>
              <overWriteSnapshots>true</overWriteSnapshots>
            </configuration>
          </execution>
        </executions>
      </plugin>
      <plugin>
        <groupId>org.mortbay.jetty</groupId>
        <artifactId>maven-jetty-plugin</artifactId>
        <version>6.1.22</version>
        <executions>
          <execution>
            <id>default-cli</id>❷
            <configuration>
              <contextPath>jackrabbit</contextPath>
              <daemon>false</daemon>
              <webApp
                ${project.build.directory}/war/jackrabbit-webapp.war
              </webApp>
            </configuration>
          </execution>
        </executions>
        <dependencies>
          <dependency>❸
            <groupId>org.apache.jackrabbit</groupId>
            <artifactId>jackrabbit-webapp</artifactId>
            <version>1.6.0</version>
            <type>war</type>
          </dependency>
          <dependency>
            <groupId>javax.jcr</groupId>
            <artifactId>jcr</artifactId>
            <version>2.0</version>
          </dependency>
        </dependencies>
      </plugin>
    </plugins>
  </build>
  <dependencies>
    <dependency>❹
      <groupId>org.apache.jackrabbit</groupId>
      <artifactId>jackrabbit-webapp</artifactId>
      <version>1.6.0</version>
      <type>war</type>
    </dependency>
  </dependencies>
</project>
```

This POM uses some advanced concepts to configure the default command-line options for both the Maven Dependency plugin and the Maven Jetty plugin. It also downloads and executes the jackrabbit-webapp.war artifact from the Central Maven repository.

❶ This particular plugin configuration configures the default command-line execution of a goal by using the default-cli identifier. This configures the default configuration that is used when a plugin goal is executed directly on the command-line (i.e. mvn dependency:copy). In this section of the POM, the Maven Dependency plugin is configured to download the Jackrabbit web application artifact and to save this file in ${project.build.directory}/war/jackrabbit-webapp.war.

❷ Again, using the identifier of default-cli specifies the configuration of the plugin when one of its goals is executed on the command-line. This particular configuration points the Jetty plugin at the downloaded jackrabbit-webapp.war file, sets the context path to jackrabbit, and sets the daemon parameter to false. Setting daemon to false will cause the Maven build to block and wait for the Jetty process.

❸ This dependencies element under the Maven Jetty plugin configuration configures the classpath for the Jetty servlet container. Without declaring these dependencies, the Jackrabbit web application would not have access to the libraries that are required for execution. Listing the jackrabbit-webapp artifact as a dependency for this plugin execution causes Maven to read the POM for this artifact and download all of the transitive dependencies of the web application artifact. This element also adds in an implementation of the Java Content Repository (JCR) API.

❸ Even though this project is nothing more than a place-holder project for the plugin configuration contained in the POM it does declare a dependency on the jackrabbit-webapp artifact it requires for execution. This particular dependency block is unnecessary, but its presence helps to declare the explicit dependency for other tools which may consume this POM.

To download the Jackrabbit web application and execute it in Jetty, copy the POM to an empty directory and run **`mvn clean dependency:copy jetty:run-war`**. Because you've configured the default command-line options in the POM, you should Maven downloading Jackrabbit, all of its dependencies, and starting the application in Jetty on port 8080.

```
$ mvn clean dependency:copy jetty:run-war
[INFO] Scanning for projects...
[INFO] ------------------------------------------------------------------------
[INFO] Building start-jackrabbit
[INFO]    task-segment: [clean, dependency:copy, jetty:run-war]
[INFO] ------------------------------------------------------------------------
[INFO] [clean:clean {execution: default-clean}]
[INFO] Deleting directory /Users/Tim/Library/Code/sonatype/maven-cookbook/mcookbook-examples/web/\
       start-jackrabbit/target
[INFO] [dependency:copy {execution: default-cli}]
[INFO] Configured Artifact: org.apache.jackrabbit:jackrabbit-webapp:1.6.0:war
[INFO] Copying jackrabbit-webapp-1.6.0.war to /Users/Tim/Library/Code/sonatype/maven-cookbook/\
       mcookbook-examples/web/start-jackrabbit/target/war/jackrabbit-webapp.war
...
[INFO] [jetty:run-war {execution: default-cli}]
[INFO] Configuring Jetty for project: start-jackrabbit
2009-11-29 18:57:26.676:INFO::Logging to STDERR via org.mortbay.log.StdErrLog
[INFO] Context path = /jackrabbit
[INFO] Tmp directory =  determined at runtime
[INFO] Web defaults = org/mortbay/jetty/webapp/webdefault.xml
[INFO] Web overrides =  none
[INFO] Starting jetty 6.1.22 ...
2009-11-29 18:57:26.811:INFO::jetty-6.1.22
2009-11-29 18:57:26.845:INFO::Extract /Users/Tim/Library/Code/sonatype/maven-cookbook/mcookbook-examples/\
          web/start-jackrabbit/target/war/jackrabbit-webapp.war to \
          /Users/Tim/Library/Code/sonatype/maven-cookbook/mcookbook-examples/web/start-jackrabbit/target/\
          work/webapp
2009-11-29 18:57:27.796:INFO::No Transaction manager found - if your webapp requires one, please configure one
29.11.2009 18:57:28 *INFO * root: Logging initialized. (LoggingServlet.java, line 87)
29.11.2009 18:57:28 *INFO * RepositoryStartupServlet: RepositoryStartupServlet initializing...
                            (RepositoryStartupServlet.java, line 235)
29.11.2009 18:57:28 *ERROR* RepositoryStartupServlet: Repository startup configuration is not valid but a
          bootstrap config is specified. (RepositoryStartupServlet.java, line 366)
29.11.2009 18:57:28 *ERROR* RepositoryStartupServlet: Either create the jackrabbit/bootstrap.properties file
          or (RepositoryStartupServlet.java, line 367)
29.11.2009 18:57:28 *ERROR* RepositoryStartupServlet: use the '/config/index.jsp' for easy configuration.
          (RepositoryStartupServlet.java, line 368)
29.11.2009 18:57:28 *ERROR* RepositoryStartupServlet: RepositoryStartupServlet initializing failed:
```

```
           javax.servlet.ServletException: Repository startup configuration is not valid.
           (RepositoryStartupServlet.java, line 245)
29.11.2009 18:57:28 *INFO * RepositoryAccessServlet: RepositoryAccessServlet initialized.
           (RepositoryAccessServlet.java, line 98)
29.11.2009 18:57:28 *INFO * SimpleWebdavServlet: resource-path-prefix = '/repository'
           (SimpleWebdavServlet.java, line 145)
29.11.2009 18:57:28 *INFO * SimpleWebdavServlet: WWW-Authenticate header = 'Basic realm="Jackrabbit Webdav
           Server"' (SimpleWebdavServlet.java, line 151)
2009-11-29 18:57:28.982:INFO::Started SelectChannelConnector@0.0.0.0:8080
[INFO] Started Jetty Server
```

Once Maven has started Jetty with the jackrabbit-webapp.war JAR, go to a web browser and open http://localhost:8080/jackrabbit. You should see the Apache Jackrabbit administrative web interface.

Note

The first time you start the Jackrabbit web application, it will print an exception stack trace as it tries to locate a Jackrabbit database. This stack trace disappears once you load the Jackrabbit administrative interface and create a new Jackrabbit database.

While this isn't the most straightforward use of a Maven POM, it does demonstrate the power of using Maven's dependency management and plugin configuration to distribute the settings necessary to execute a web application. If you used such a mechanism to distribute your own web applications, you could configure a web application and all of the information required to download your web application, it's dependencies, the servlet container, and any configuration needed to run your application.

Chapter 7. Unit Testing with Maven

7.1. Introduction

Maven can run Unit Tests, this chapter shows you how.

7.2. Running JUnit Tests

7.2.1. Task

You need to run all JUnit tests in a given project.

7.2.2. Action

To execute all of the unit tests in a project, include JUnit as a test scoped dependency in your project's pom.xml:

```
<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/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.sonatype.mcookbook</groupId>
  <artifactId>junit-tests</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>junit-tests</name>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.5</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

Your project should then store unit classes in the default location of `${basedir}/src/test/java`. The Maven Surefire plugin will scan these directory for JUnit tests. In this example, a component named SeriousComponent has a corresponding JUnit test named SeriousComponentTest.

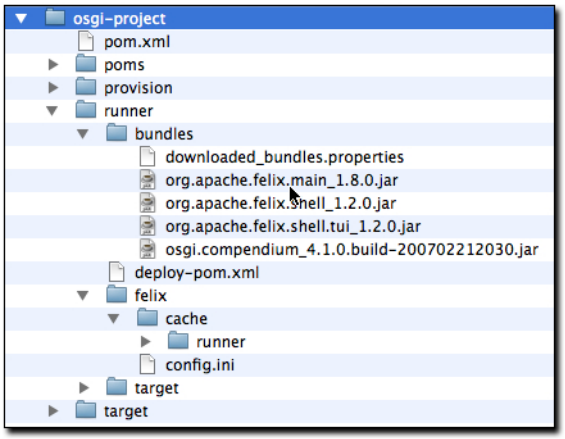

Figure 7.1. Project Structure for Unit Tests

In this example, the simple component being tested is named SeriousComponent. SeriousComponent contains a single static method to be tested.

Example 7.1. SeriousComponent class

```
package org.sonatype.mcookbook;

public class SeriousComponent {

 /**
  * This function tests the seriousness of a String.
  * Returns false if the string contains the word
  * "FUNNY", returns true otherwise.
  */
  public static boolean testSeriousness( String text ) {
    return !text.toUpperCase().contains( "FUNNY" );
  }
}
```

This class is tested by a simple JUnit test - SeriousComponentTest shown in the following exmple:

Example 7.2. JUnit test for SeriousComponent

```
package org.sonatype.mcookbook;

import junit.framework.TestCase;

public class SeriousComponentTest extends TestCase {

        public SeriousComponentTest(String name) {
                super( name );
        }

        public void testSeriousness() throws Exception {
                assertTrue( SeriousComponent.testSeriousness( "SAD" ) );
                assertTrue( SeriousComponent.testSeriousness( "SERIOUS" ) );
                assertTrue( SeriousComponent.testSeriousness( "CRAZY" ) );
                assertTrue( !SeriousComponent.testSeriousness( "FUNNY" ) );
        }
}
```

To execute your unit test, you don't need to do anything. Maven's default settings are to scan `${basedir}/src/test/java` for unit tests matching the pattern `*Test.java`. To run your unit test specify the test phase of the default Maven lifecycle and run **`mvn test`**.

```
$ mvn test
[INFO] Scanning for projects...
[INFO] ------------------------------------------------------------------------
[INFO] Building junit-tests
[INFO]    task-segment: [test]
[INFO] ------------------------------------------------------------------------
[INFO] [resources:resources {execution: default-resources}]
[WARNING] Using platform encoding (MacRoman actually) to copy filtered
resources, i.e. build is platform dependent!
[INFO] Copying 0 resource
[INFO] [compiler:compile {execution: default-compile}]
[INFO] Compiling 1 source file to /Users/Tim/Library/Code/sonatype/
maven-cookbook/mcookbook-examples/unit/junit-tests/target/classes
[INFO] [resources:testResources {execution: default-testResources}]
[WARNING] Using platform encoding (MacRoman actually) to copy filtered
resources, i.e. build is platform dependent!
[INFO] Copying 0 resource
[INFO] [compiler:testCompile {execution: default-testCompile}]
[INFO] Compiling 1 source file to /Users/Tim/Library/Code/sonatype/
```

```
maven-cookbook/mcookbook-examples/unit/junit-tests/target/test-classes
[INFO] [surefire:test {execution: default-test}]
[INFO] Surefire report directory: /Users/Tim/Library/Code/sonatype/
maven-cookbook/mcookbook-examples/unit/junit-tests/target/surefire-reports

-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running org.sonatype.mcookbook.SeriousComponentTest
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.065 sec

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 5 seconds
[INFO] Finished at: Wed Nov 25 14:42:39 CST 2009
[INFO] Final Memory: 17M/80M
[INFO] ------------------------------------------------------------------------
```

7.2.3. Detail

The test results will be made available in `${basedir}/target/surefire-reports` as both an XML file and a text file.

Figure 7.2. Project Structure for JUnit Test Results

The TEST-org.sonatype.mcookbook.SeriousComponentTest.xml contains an XML document describing the environment and state of the JVM in addition to data about the test cases which have been executed, and the org.sonatype.mcookbook.SeriousComponentTest.txt file a summary of a successful test or output that contains stack traces generated by a test failure.

If your unit test passes, the org.sonatypemcookbook.SeriousComponentTest.txt will contain the following output:

```
$ cd target/surefire-reports/
$ more org.sonatype.mcookbook.SeriousComponentTest.txt
-------------------------------------------------------------------------------
Test set: org.sonatype.mcookbook.SeriousComponentTest
-------------------------------------------------------------------------------
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.065 sec
```

If the unit test fails the same file will contain a stack trace that points to a failed assertion on a specific line of the unit test as shown below:

```
-------------------------------------------------------------------------------
Test set: org.sonatype.mcookbook.SeriousComponentTest
-------------------------------------------------------------------------------
Tests run: 1, Failures: 1, Errors: 0, Skipped: 0, Time elapsed: 0.062 sec
<<< FAILURE!
testSeriousness(org.sonatype.mcookbook.SeriousComponentTest)  Time elapsed:
0.017 sec  <<< FAILURE!
junit.framework.AssertionFailedError: null
        at junit.framework.Assert.fail(Assert.java:47)
        at junit.framework.Assert.assertTrue(Assert.java:20)
        at junit.framework.Assert.assertTrue(Assert.java:27)
        at org.sonatype.mcookbook.SeriousComponentTest.testSeriousness(
SeriousComponentTest.java:15)
....
```

7.3. Running TestNG Tests

7.3.1. Task

You need to run all TestNG tests in a given project.

7.3.2. Action

To use TestNG, you will need to add TestNG as a dependency in your project's POM. Since the TestNG unit test shown in this section uses Java 5 annotation, you will also need to configure the Maven Compiler plugin to target Java 5. The following POM shows the minimum required configuration for running TestNG tests with Java 5 annotations.

```
<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/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.sonatype.mcookbook</groupId>
  <artifactId>testng-tests</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>testng-tests</name>
  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <configuration>
          <source>1.5</source>
          <target>1.5</target>
        </configuration>
      </plugin>
    </plugins>
  </build>
  <dependencies>
    <dependency>
      <groupId>org.testng</groupId>
      <artifactId>testng</artifactId>
      <version>5.10</version>
      <classifier>jdk15</classifier>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

You would then put your TestNG unit test classses in the `${basedir}/src/test/java` directory. The following TestNG test uses Java 5 annotations to mark a class and a method as a TestNG test. The following TestNG unit test duplicates the JUnit test shown in Section 7.2, “Running JUnit Tests”, and it tests the SeriousComponent class.

Example 7.3. TestNG Test for the SeriousComponent

```
package org.sonatype.mcookbook;

import org.testng.annotations.Test;

@Test
public class SeriousComponentTest {

  @Test
  public void testSeriousness() throws Exception {
    assert SeriousComponent.testSeriousness("SAD");
    assert SeriousComponent.testSeriousness("SERIOUS");
    assert SeriousComponent.testSeriousness("CRAZY");
    assert !SeriousComponent.testSeriousness("FUNNY");
```

```
  }
}
```

There is no other configuration necessary to have Maven execute any TestNG tests it locates under `${basedir}/src/test/java` which match the pattern `*Test.java`. Running **`mvn test`** will execute your TestNG unit tests.

```
$ mvn test
[INFO] Scanning for projects...
[INFO] ------------------------------------------------------------------------
[INFO] Building testng-tests
[INFO]    task-segment: [test]
[INFO] ------------------------------------------------------------------------
[INFO] [resources:resources {execution: default-resources}]
[WARNING] Using platform encoding (MacRoman actually) to copy filtered
resources, i.e. build is platform dependent!
[INFO] Copying 0 resource
[INFO] [compiler:compile {execution: default-compile}]
[INFO] Compiling 1 source file to /Users/Tim/Library/Code/sonatype/
maven-cookbook/mcookbook-examples/unit/testng-tests/target/classes
[INFO] [resources:testResources {execution: default-testResources}]
[WARNING] Using platform encoding (MacRoman actually) to copy filtered
resources, i.e. build is platform dependent!
[INFO] Copying 0 resource
[INFO] [compiler:testCompile {execution: default-testCompile}]
[INFO] Compiling 1 source file to /Users/Tim/Library/Code/sonatype/
maven-cookbook/mcookbook-examples/unit/testng-tests/target/test-classes
[INFO] [surefire:test {execution: default-test}]
[INFO] Surefire report directory: /Users/Tim/Library/Code/sonatype/
maven-cookbook/mcookbook-examples/unit/testng-tests/target/surefire-reports

-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running TestSuite
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.292 sec

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 4 seconds
[INFO] Finished at: Thu Nov 26 08:24:02 CST 2009
[INFO] Final Memory: 17M/80M
[INFO] ------------------------------------------------------------------------
```

7.3.3. Detail

Once the TestNG unit tests have been executed, you will have a TestNG HTML report under `${basedir}/target/surfire-reports/index.html`. To view a report which displays statistics about which tests failed and which tests passed, load this HTML page in a web browser.

Figure 7.3. Project Structure for TestNG Test Results

7.4. Running Specific TestNG Test Groups

7.4.1. Task

You want to configure the Maven Surefire plugin to run specific tests that are defined in a TestNG Test Suite XML descriptor.

7.4.2. Action

Consider a TestNG test class which contains five methods with the @Test annotation. Each method also configures the groups parameter. Four methods are in the "unit" group and one method is in the "integration" group. This might be a common situation for a project that contains different kinds of tests that are to be run during different times of the software development lifecycle. Very often, there is a set of tests that should be run every time a programmer makes a change to source code. There may also be a certain kind of test that can only be executed on a specific integration and testing machine, and there can be other tests which require too much time and resources to be associated with a developer's daily development cycle.

Here is the SeriousComponentTest with five annotated test methods:

```
package org.sonatype.mcookbook;

import org.testng.annotations.Test;

public class SeriousComponentTest {

  @Test(groups={"unit"})
  public void testSad() throws Exception {
    assert SeriousComponent.testSeriousness("SAD");
  }

  @Test(groups={"unit"})
  public void testSerious() throws Exception {
    assert SeriousComponent.testSeriousness("SERIOUS");
  }

  @Test(groups={"unit"})
  public void testCrazy() throws Exception {
    assert SeriousComponent.testSeriousness("CRAZY");
  }

  @Test(groups={"unit"})
  public void testFunny() throws Exception {
    assert !SeriousComponent.testSeriousness("FUNNY");
  }

  @Test(groups={"integration"})
  public void testLargeFile() throws Exception {
    String text = "TEST";
    // Imagine that this method contained some serious
    // code that loaded a 100k line text file and
    // tested each line.
    assert SeriousComponent.testSeriousness(text);
  }
}
```

The first four methods are executed quickly as unit tests, but the last method requires more time to execute, and if going to be run periodically by a continuous integration server like Hudson. To make the development cycle more efficient for your developers, you would add the following Maven Surefire plugin configuration to your project's POM. Specifying the groups parameter allows you to select one or more, comma-separate group names which will be executed by TestNG.

```
<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/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.sonatype.mcookbook</groupId>
  <artifactId>testng-groups</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>testng-groups</name>
  <build>
    <plugins>
```

```
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <configuration>
          <source>1.5</source>
          <target>1.5</target>
        </configuration>
      </plugin>
      <plugin>
       <groupId>org.apache.maven.plugins</groupId>
       <artifactId>maven-surefire-plugin</artifactId>
       <configuration>
         <groups>unit</groups>
       </configuration>
      </plugin>
    </plugins>
  </build>
  <dependencies>
    <dependency>
      <groupId>org.testng</groupId>
      <artifactId>testng</artifactId>
      <version>5.10</version>
      <classifier>jdk15</classifier>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

When you execute **`mvn test`**, you will see that the Maven Surefire plugin will only execute four of the five tests in SeriousComponentTest.

7.4.3. Detail

If you wanted to run all tests in the SeriousComponentTest, you could configure your POM to have the following Maven Surefire configuration:

```
      <plugin>
       <groupId>org.apache.maven.plugins</groupId>
       <artifactId>maven-surefire-plugin</artifactId>
       <configuration>
         <groups>unit, integration</groups>
       </configuration>
      </plugin>
```

Alternatively, you could just define a dependency on the "unit" group withint the SeriousComponentTest class on the testLargeFile() method. Here's the annotation syntax you would use to do this:

```
  @Test(groups={"integration"}, dependsOnGroups={"unit"})
  public void testLargeFile() throws Exception {
    String text = "TEST";
    // Imagine that this method contained some serious
    // code that loaded a 100k line text file and
    // tested each line.
    assert SeriousComponent.testSeriousness(text);
  }
```

Then, in your project's POM, you would simply configure TestNG to run the "integration" group. TestNG would take note of the dependency defined in the test class and execute the "unit" group before the "integration" group.

```
      <plugin>
       <groupId>org.apache.maven.plugins</groupId>
       <artifactId>maven-surefire-plugin</artifactId>
       <configuration>
         <groups>integration</groups>
```

```
      </configuration>
    </plugin>
```

TestNG supports a very rich set of configuration possibilities for defining groups. You can have groups which depend on other groups, groups can be defined in Test Suite XML files, and you can match groups based on regular expressions to allow for platform specific customizations. Getting into the details of TestNG groups is well beyond the scope of this book, if you would like more information about TestNG, please see the TestNG project documentation here: http://testng.org/doc/documentation-main.html#test-groups.

7.5. Configuring TestNG Tests

7.5.1. Task

You need to pass configuration parameters to TestNG unit tests.

7.5.2. Action

Consider the following TestNG unit test which declares the parameter "databaseHostname" using the @Parameters annotation.

```
package org.sonatype.mcookbook;

import org.testng.annotations.Parameters;
import org.testng.annotations.Test;

@Test
public class SeriousComponentTest {

  @Parameters({"databaseHostname"})
  @Test
  public void testSeriousness(String databaseHostname) throws Exception {
    assert SeriousComponent.testSeriousness("SAD");
    assert SeriousComponent.testSeriousness("SERIOUS");
    assert SeriousComponent.testSeriousness("CRAZY");
    assert !SeriousComponent.testSeriousness("FUNNY");
    System.out.println( "Database Hostname: " + databaseHostname );
  }
}
```

This "databaseHostname" parameter can be configured by using the systemProperties configuration parameter of the Maven Surefire plugin. The following POM passes in a value of "testDb01" for the "databaseHostname" test parameter.

```
<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/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.sonatype.mcookbook</groupId>
  <artifactId>testng-config</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>testng-config</name>
  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <configuration>
          <source>1.5</source>
          <target>1.5</target>
        </configuration>
      </plugin>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
```

```
        <artifactId>maven-surefire-plugin</artifactId>
        <configuration>
          <systemProperties>
            <property>
              <name>databaseHostname</name>
              <value>testDb01</value>
            </property>
          </systemProperties>
        </configuration>
      </plugin>
    </plugins>
  </build>
  <dependencies>
    <dependency>
      <groupId>org.testng</groupId>
      <artifactId>testng</artifactId>
      <version>5.10</version>
      <classifier>jdk15</classifier>
      <scope>test</scope>
    </dependency>
  </dependencies>

</project>
```

Running the test phase will execute the unit test passing the value from the POM to the unit test as a system property.

```
$ mvn test
[INFO] Scanning for projects...
[INFO] ------------------------------------------------------------------------
[INFO] Building testng-config
[INFO]    task-segment: [test]
[INFO] ------------------------------------------------------------------------
[INFO] [resources:resources {execution: default-resources}]
[WARNING] Using platform encoding (MacRoman actually) to copy filtered
resources, i.e. build is platform dependent!
[INFO] Copying 0 resource
[INFO] [compiler:compile {execution: default-compile}]
[INFO] Nothing to compile - all classes are up to date
[INFO] [resources:testResources {execution: default-testResources}]
[WARNING] Using platform encoding (MacRoman actually) to copy filtered
resources, i.e. build is platform dependent!
[INFO] Copying 0 resource
[INFO] [compiler:testCompile {execution: default-testCompile}]
[INFO] Compiling 1 source file to /Users/Tim/Library/Code/sonatype/
maven-cookbook/mcookbook-examples/unit/testng-config/target/test-classes
[INFO] [surefire:test {execution: default-test}]
[INFO] Surefire report directory: /Users/Tim/Library/Code/sonatype/
maven-cookbook/mcookbook-examples/unit/testng-config/target/surefire-reports

-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running TestSuite
Database Hostname: testDb01
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.358 sec

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 4 seconds
[INFO] Finished at: Thu Nov 26 13:32:44 CST 2009
[INFO] Final Memory: 17M/80M
[INFO] ------------------------------------------------------------------------
```

7.5.3. Detail

For more information about TestNG parameters, see the TestNG project documentation here: http://testng.org/doc/documentation-main.html#parameters.

7.6. Running TestNG Tests in Parallel

7.6.1. Task

You would like to run your unit tests in parallel to speed up your builds.

7.6.2. Action

To illustrate executing tests in parallel, we can create a component to be tests that sleeps for 5 seconds. The following listing is the SeriousComponent.java class which would be stored in `${basedir}/src/main/java` under the org.sonatype.mcookbook package:

```
package org.sonatype.mcookbook;

public class SeriousComponent {

 /**
  * This function tests the seriousness of a String.
  * Returns false if the string contains the word
  * "FUNNY", returns truje otherwise.
  */
  public static boolean testSeriousness( String text ) {
    try {
      Thread.sleep( 5000 );
    } catch (InterruptedException e) {
    }
    return !text.toUpperCase().contains( "FUNNY" );
  }
}
```

This class has a single static method which sleeps for five seconds and then tests a string for the presence of the word "funny". To test this class, there is a SeriousComponentTest.java class in `${basedir}/src/test/java` under the org.sonatype.mcookbook package which contains the following four test methods:

```
package org.sonatype.mcookbook;

import org.testng.annotations.Test;

@Test
public class SeriousComponentTest {

  @Test
  public void testSad() throws Exception {
    assert SeriousComponent.testSeriousness("SAD");
  }

  @Test
  public void testSerious() throws Exception {
    assert SeriousComponent.testSeriousness("SERIOUS");
  }

  @Test
  public void testCrazy() throws Exception {
    assert SeriousComponent.testSeriousness("CRAZY");
  }

  @Test
  public void testFunny() throws Exception {
```

```
    assert !SeriousComponent.testSeriousness("FUNNY");
  }
}
```

If you ran this test in a single thread, it would take approximately 20 seconds to execute. To speed up your build, configure Maven to execute each test method in parallel and set the available thread count for parallel test execution to 4. To do this, add the following plugin configuration to your project's POM.

```
<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/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.sonatype.mcookbook</groupId>
  <artifactId>testng-parallel</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>testng-tests</name>
  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <configuration>
          <source>1.5</source>
          <target>1.5</target>
        </configuration>
      </plugin>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-surefire-plugin</artifactId>
        <configuration>
          <parallel>methods</parallel>
          <threadCount>4</threadCount>
        </configuration>
      </plugin>
    </plugins>
  </build>
  <dependencies>
    <dependency>
      <groupId>org.testng</groupId>
      <artifactId>testng</artifactId>
      <version>5.10</version>
      <classifier>jdk15</classifier>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

Execute the TestNG test in parallel by running **`mvn test`** and take note of the time it takes to execute all four tests.

```
$ mvn test
[INFO] Scanning for projects...
[INFO] ------------------------------------------------------------------------
[INFO] Building testng-tests
[INFO]    task-segment: [test]
[INFO] ------------------------------------------------------------------------
[INFO] [resources:resources {execution: default-resources}]
[WARNING] Using platform encoding (MacRoman actually) to copy filtered
resources, i.e. build is platform dependent!
[INFO] Copying 0 resource
[INFO] [compiler:compile {execution: default-compile}]
[INFO] Nothing to compile - all classes are up to date
[INFO] [resources:testResources {execution: default-testResources}]
[WARNING] Using platform encoding (MacRoman actually) to copy filtered
resources, i.e. build is platform dependent!
[INFO] Copying 0 resource
```

```
[INFO] [compiler:testCompile {execution: default-testCompile}]
[INFO] Nothing to compile - all classes are up to date
[INFO] [surefire:test {execution: default-test}]
[INFO] Surefire report directory: /Users/Tim/Library/Code/sonatype/
maven-cookbook/mcookbook-examples/unit/testng-parallel/target/surefire-reports

-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running TestSuite
Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 5.342 sec

Results :

Tests run: 4, Failures: 0, Errors: 0, Skipped: 0
```

You can see that the execution of four tests took a total of 5.342 sec. This is a clear indication that the tests were being run in parallel given the fact that the function being tested has a 5 second sleep built-in. Now, change the parallel configuration parameter to "classes" and rerun the test. Find the Maven Surefire plugin configuration in the project's POM and change the configuration to this:

```
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-surefire-plugin</artifactId>
        <configuration>
          <parallel>classes</parallel>
          <threadCount>4</threadCount>
        </configuration>
      </plugin>
```

Now, run **`mvn test`** and notice how long it takes to run the unit tests - 20 seconds. When you run the tests in parallel using classes, TestNG is going to create an independent test execution thread for each class. The four test methods in SeriousComponentTest are going to be executed in one, single thread instead of the four simultaneous threads that were used when the parallel configuration parameter was set to "methods".

```
$ mvn test
[INFO] Scanning for projects...
[INFO] ------------------------------------------------------------------------
[INFO] Building testng-tests
[INFO]    task-segment: [test]
[INFO] ------------------------------------------------------------------------
[INFO] [resources:resources {execution: default-resources}]
[WARNING] Using platform encoding (MacRoman actually) to copy filtered
resources, i.e. build is platform dependent!
[INFO] Copying 0 resource
[INFO] [compiler:compile {execution: default-compile}]
[INFO] Nothing to compile - all classes are up to date
[INFO] [resources:testResources {execution: default-testResources}]
[WARNING] Using platform encoding (MacRoman actually) to copy filtered
resources, i.e. build is platform dependent!
[INFO] Copying 0 resource
[INFO] [compiler:testCompile {execution: default-testCompile}]
[INFO] Nothing to compile - all classes are up to date
[INFO] [surefire:test {execution: default-test}]
[INFO] Surefire report directory: /Users/Tim/Library/Code/sonatype/
maven-cookbook/mcookbook-examples/unit/testng-parallel/target/surefire-reports

-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running TestSuite
Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 20.411 sec

Results :

Tests run: 4, Failures: 0, Errors: 0, Skipped: 0
```

7.6.3. Detail

If waiting around for your unit tests is eating up too much of your time, this is a great way to make your builds a bit faster.

You use the following values in the parallel configuration parmeter:

classes
: Each test class will be executed in an independent execution thread.

methods
: Each test method will be run as an independent test in a separate thread. Note that if one method states and explicit dependency on another method, TestNG will note the dependency and execute both methods in the same Thread.

tests
: If you use a TestNG Suite XML file and you list a test element that groups a number of related tests, listing test in the parallel configuration element will configure TestNG to use a separate thread for each <test> element. This option is only relevant if you have configured the Maven Surefire plugin to use a custom TestNG Suite XML file with one or more <test> elements.

7.7. Skipping Unit Tests

7.7.1. Task

You need to skip unit tests in a Maven build.

7.7.2. Action

To skip unit tests in a Maven build, pass a value of true to the parameter maven.test.skip to Maven on the command line.

```
$ mvn install -Dmaven.test.skip=true
[INFO] Scanning for projects...
[INFO] ------------------------------------------------------------------------
[INFO] Building testng-groups
[INFO]    task-segment: [install]
[INFO] ------------------------------------------------------------------------
[INFO] [resources:resources {execution: default-resources}]
[WARNING] Using platform encoding (MacRoman actually) to copy filtered
resources, i.e. build is platform dependent!
[INFO] Copying 0 resource
[INFO] [compiler:compile {execution: default-compile}]
[INFO] Nothing to compile - all classes are up to date
[INFO] [resources:testResources {execution: default-testResources}]
[WARNING] Using platform encoding (MacRoman actually) to copy filtered
resources, i.e. build is platform dependent!
[INFO] Copying 0 resource
[INFO] [compiler:testCompile {execution: default-testCompile}]
[INFO] Not compiling test sources
[INFO] [surefire:test {execution: default-test}]
[INFO] Tests are skipped.
[INFO] [jar:jar {execution: default-jar}]
[INFO] Building jar: /Users/Tim/Library/Code/sonatype/maven-cookbook/
mcookbook-examples/unit/testng-groups/target/testng-groups-1.0-SNAPSHOT.jar
[INFO] [install:install {execution: default-install}]
[INFO] Installing /Users/Tim/Library/Code/sonatype/maven-cookbook/
mcookbook-examples/unit/testng-groups/target/testng-groups-1.0-SNAPSHOT.jar
to /Users/Tim/.m2/repository/org/sonatype/mcookbook/testng-groups/
1.0-SNAPSHOT/testng-groups-1.0-SNAPSHOT.jar
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 4 seconds
[INFO] Finished at: Thu Nov 26 12:32:14 CST 2009
[INFO] Final Memory: 15M/80M
```

```
[INFO] ------------------------------------------------------------------------
```

7.7.3. Detail

If you need to configure a Maven build to skip tests, you would add the following Maven Surefire configuration.

```
     <plugin>
       <groupId>org.apache.maven.plugins</groupId>
       <artifactId>maven-surefire-plugin</artifactId>
       <configuration>
         <skip>true</skip>
       </configuration>
      </plugin>
```

7.8. Running a Single Unit Test

7.8.1. Task

You want to run a single test class from a Maven project.

7.8.2. Action

To run a specific unit test in a Maven build, pass the name of the test class to the parameter test on the command line.

```
$ mvn test -Dtest=SeriousComponentTest
[INFO] Scanning for projects...
[INFO] ------------------------------------------------------------------------
[INFO] Building testng-groups
[INFO]    task-segment: [test]
[INFO] ------------------------------------------------------------------------
[INFO] [resources:resources {execution: default-resources}]
[WARNING] Using platform encoding (MacRoman actually) to copy filtered
resources, i.e. build is platform dependent!
[INFO] Copying 0 resource
[INFO] [compiler:compile {execution: default-compile}]
[INFO] Nothing to compile - all classes are up to date
[INFO] [resources:testResources {execution: default-testResources}]
[WARNING] Using platform encoding (MacRoman actually) to copy filtered
resources, i.e. build is platform dependent!
[INFO] Copying 0 resource
[INFO] [compiler:testCompile {execution: default-testCompile}]
[INFO] Nothing to compile - all classes are up to date
[INFO] [surefire:test {execution: default-test}]
[INFO] Surefire report directory: /Users/Tim/Library/Code/sonatype/
maven-cookbook/mcookbook-examples/unit/testng-groups/target/surefire-reports

-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running TestSuite
Tests run: 5, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.466 sec

Results :

Tests run: 5, Failures: 0, Errors: 0, Skipped: 0

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 5 seconds
[INFO] Finished at: Thu Nov 26 13:01:58 CST 2009
[INFO] Final Memory: 11M/80M
[INFO] ------------------------------------------------------------------------
```

Chapter 8. Integration Testing with Maven

8.1. Introduction

This chapter covers integration testing with Maven.

8.2. Running a Selenium Test

8.2.1. Task

You need to write a Selenium test to test a web application or a web site.

8.2.2. Action

Selenium is a very straightforward way to test a web application, you can write a unit test in any number of languages and then use the Selenium Remote Control to automate a browser and perform assertions about the presence of text or UI elements in a page. This recipe uses a TestNG test which includes the appropriate Selenium code to test a well known web site: http://www.twitter.com. In a Maven project devoted to integration testing, the following test class connects to a Twitter user page and verifies the correct user name.

Example 8.1. TwitterTest a Selenium Test written in TestNG

```
package org.sonatype.mcookbook;

import org.testng.annotations.AfterClass;
import org.testng.annotations.BeforeClass;
import org.testng.annotations.Parameters;
import org.testng.annotations.Test;

import com.thoughtworks.selenium.DefaultSelenium;
import com.thoughtworks.selenium.Selenium;
import com.thoughtworks.selenium.SeleniumException;

public class TwitterTest {

  private Selenium selenium;

  @BeforeClass
  public void startSelenium() {
    this.selenium = new DefaultSelenium("localhost", 4444, "*safari",
        "http://www.twitter.com");
    this.selenium.start();
  }

  @Test
  @Parameters( { "user", "name" })
  public void testTwitter(String user, String name) {
    try {
      selenium.open("http://www.twitter.com/" + user);
      selenium.waitForPageToLoad("3000");
      assert selenium.isTextPresent(name);

    } catch (SeleniumException e) {
      throw e;
    }
  }

  @AfterClass(alwaysRun = true)
  public void stopSelenium() {
    this.selenium.stop();
  }
```

```
}
```

The following POM contains the following configuration to configure the Selenium server and the Selenium integration tests:

- The Maven Surefire plugin is configured to skip tests. This will prevent the Surefire plugin from executing any unit tests during the test phase.
- An execution of the Surefire plugin is configured not to skip tests during the integration-test phase. This configuration coupled with the previous configuration simply moves the execution of the tests into the integration-test phase of the lifecycle.
- The user and name properties that the testTwitter() method relies upon are configured via the systemProperties configuration parameter of the Maven Surefire plugin.
- The Selenium Maven plugin's start-server goal is configured to run in the pre-integration-test phase. This has the effect of starting Selenium before the Surefire plugin executes the integration tests. No configuration options are passed to the start-server goal so Selenium will be started on a default port of 4444.
- A dependency on selenium-java-client-driver exposes the Selenium API to the test cases and allows the Surefire test to interact with the Selenium server.

Example 8.2. POM configuring SureFire to Execute Selenium Remote Control

```
<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/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.sonatype.mcookbook</groupId>
  <artifactId>selenium-reddit</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>selenium-reddit</name>
  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <configuration>
          <source>1.5</source>
          <target>1.5</target>
        </configuration>
      </plugin>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-surefire-plugin</artifactId>
        <configuration>
          <skip>true</skip>
          <systemProperties>
            <property>
              <name>user</name>
              <value>tobrien</value>
            </property>
            <property>
              <name>name</name>
              <value>Tim O'Brien</value>
            </property>
          </systemProperties>
        </configuration>
        <executions>
          <execution>
            <phase>integration-test</phase>
            <goals>
              <goal>test</goal>
            </goals>
            <configuration>
              <skip>false</skip>
```

```
            </configuration>
          </execution>
        </executions>
      </plugin>
      <plugin>
        <groupId>org.codehaus.mojo</groupId>
        <artifactId>selenium-maven-plugin</artifactId>
        <executions>
          <execution>
            <phase>pre-integration-test</phase>
            <goals>
              <goal>start-server</goal>
            </goals>
            <configuration>
              <background>true</background>
            </configuration>
          </execution>
        </executions>
      </plugin>
    </plugins>
  </build>
  <dependencies>
    <dependency>
      <groupId>org.testng</groupId>
      <artifactId>testng</artifactId>
      <version>5.10</version>
      <classifier>jdk15</classifier>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>org.seleniumhq.selenium.client-drivers</groupId>
      <artifactId>selenium-java-client-driver</artifactId>
      <version>1.0.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

If you run Maven to the integration-test phase, Maven will start the Selenium server before it executes the integration-test phase. In the output shown below, the section that starts with selenium:start-server is the pre-integration-test execution of the start-server goal in the Selenium Maven plugin. This is followed by an invocation of the test goal from the Maven Surefire plugin which will execute the TestNG tests which rely on the Selenium server.

```
$ mvn integration-test
[INFO] Scanning for projects...
[INFO] ------------------------------------------------------------------------
[INFO] Building selenium-reddit
[INFO]    task-segment: [integration-test]
[INFO] ------------------------------------------------------------------------
[INFO] [resources:resources {execution: default-resources}]
[WARNING] Using platform encoding (MacRoman actually) to copy filtered
resources, i.e. build is platform dependent!
[INFO] Copying 0 resource
[INFO] [compiler:compile {execution: default-compile}]
[INFO] Nothing to compile - all classes are up to date
[INFO] [resources:testResources {execution: default-testResources}]
[WARNING] Using platform encoding (MacRoman actually) to copy filtered
resources, i.e. build is platform dependent!
[INFO] Copying 0 resource
[INFO] [compiler:testCompile {execution: default-testCompile}]
[INFO] Compiling 1 source file to /Users/Tim/Library/Code/sonatype/
maven-cookbook/mcookbook-examples/integrate/selenium-reddit/
target/test-classes
[INFO] [surefire:test {execution: default-test}]
[INFO] Tests are skipped.
[INFO] [jar:jar {execution: default-jar}]
[INFO] Building jar: /Users/Tim/Library/Code/sonatype/maven-cookbook/
```

```
mcookbook-examples/integrate/selenium-reddit/
target/selenium-reddit-1.0-SNAPSHOT.jar
[INFO] [selenium:start-server {execution: default}]
Created dir: /Users/Tim/Library/Code/sonatype/maven-cookbook/
mcookbook-examples/integrate/selenium-reddit/target/selenium
Launching Selenium Server
Waiting for Selenium Server...
[INFO] User extensions: /Users/Tim/Library/Code/sonatype/maven-cookbook/
mcookbook-examples/integrate/selenium-reddit/
target/selenium/user-extensions.js
23:41:46,726 INFO  [SeleniumServer] Java: Apple Inc. 14.1-b02-90
23:41:46,727 INFO  [SeleniumServer] OS: Mac OS X 10.6.1 x86_64
23:41:46,735 INFO  [SeleniumServer] v1.0.1 [2697], with Core
23:41:46,829 INFO  [HttpServer] Version Jetty/5.1.x
23:41:46,830 INFO  [Container] Started HttpContext[/selenium-server/driver,
/selenium-server/driver]
23:41:46,832 INFO  [Container] Started HttpContext[/selenium-server,
/selenium-server]
23:41:46,832 INFO  [Container] Started HttpContext[/,/]
23:41:46,846 INFO  [SocketListener] Started SocketListener on 0.0.0.0:4444
23:41:46,846 INFO  [Container] Started org.mortbay.jetty.Server@4a4e79f1
23:41:47,262 INFO  [Credential] Checking Resource aliases
Selenium Server started
[INFO] [surefire:test {execution: default}]
[INFO] Surefire report directory: /Users/Tim/Library/Code/sonatype/
maven-cookbook/mcookbook-examples/integrate/selenium-reddit/
target/surefire-reports

-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running TestSuite
23:41:48,178 INFO  [org.mortbay.util.Credential] Checking Resource aliases
23:41:48,184 INFO  [org.openqa.selenium.server.SeleniumDriverResourceHandler]
Command request: getNewBrowserSession[*safari, http://www.twitter.com, ] on
session null
23:41:48,187 INFO  [org.openqa.selenium.server.BrowserSessionFactory]
creating new remote session
23:41:48,436 INFO  [org.openqa.selenium.server.BrowserSessionFactory]
Allocated session 2a66870a3c894a60a19ef6f2e7a1dc74 for
http://www.twitter.com, launching...
23:41:48,519 INFO  [org.openqa.selenium.server.browserlaunchers.
SafariCustomProfileLauncher] Launching Safari to visit...
23:41:53,251 INFO  [org.openqa.selenium.server.SeleniumDriverResourceHandler]
Got result: OK, on session
23:41:53,265 INFO  [org.openqa.selenium.server.SeleniumDriverResourceHandler]
Command request: open[http://www.twitter.com/tobrien, ] on session
23:41:56,231 INFO  [org.openqa.selenium.server.SeleniumDriverResourceHandler]
Got result: OK on session
23:41:56,256 INFO  [org.openqa.selenium.server.SeleniumDriverResourceHandler]
Command request: waitForPageToLoad[3000, ] on session
23:41:56,270 INFO  [org.openqa.selenium.server.SeleniumDriverResourceHandler]
Got result: OK on session
23:41:56,284 INFO  [org.openqa.selenium.server.SeleniumDriverResourceHandler]
Command request: isTextPresent[Tim O'Brien, ] on session
23:41:56,343 INFO  [org.openqa.selenium.server.SeleniumDriverResourceHandler]
Got result: OK,true on session
23:41:56,352 INFO  [org.openqa.selenium.server.SeleniumDriverResourceHandler]
Command request: testComplete[, ] on session
23:41:56,370 INFO  [org.openqa.selenium.server.SeleniumDriverResourceHandler]
Got result: OK on session
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 8.609 sec

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

[INFO] ------------------------------------------------------------------------
```

```
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 18 seconds
[INFO] Finished at: Fri Nov 27 23:41:56 CST 2009
[INFO] Final Memory: 32M/80M
[INFO] ------------------------------------------------------------------------
```

8.3. Running Integration Tests Against a Servlet Container

8.3.1. Task

You are testing a web application and you need to start a servlet container prior to running integration tests.

8.3.2. Action

Use the Maven Jetty Plugin to start an instance of a server prior to running your integration tests. Assume that you are writing integration tests to test the sample web application project that was introduced in Section 6.1, "Running a Web Application in a Servlet Container". If you have some Selenium tests for the web application, you can start an instance of Jetty running the web application as a daemon in the pre-integration-test phase and stop the instance in the post-integration-test phase.

Your web application is in the org.sonatype.mcookbook:sample-web project and your integration tests are in the org.sonatype-mcookbook:sample-web-it project. You can configure the Maven Dependency plugin to copy the WAR to the sample-web-it project during the package phase, then during the pre-integation-test phase you will configure the Maven Jetty plugin to start an instance of Jetty to run the sample-web.war as well as to start the Selenium server. The Maven Surefire plugin will then execute the unit tests, and after the tests are complete the Jetty server will be stopped in the post-integration-test phase.

Our TestNG integration tests is as follows. It is going to connect to the default Jetty host and port of localhost:8080, and submit a form that calculates the tenth number in the Fibonacci sequence (F_{10}). This integration test is stored in `${basedir}/src/test/java`.

```
package org.sonatype.mcookbook;

import org.testng.annotations.AfterClass;
import org.testng.annotations.BeforeClass;
import org.testng.annotations.Parameters;
import org.testng.annotations.Test;

import com.thoughtworks.selenium.DefaultSelenium;
import com.thoughtworks.selenium.Selenium;
import com.thoughtworks.selenium.SeleniumException;

public class FibTest {

private Selenium selenium;

  @BeforeClass
  public void startSelenium() {
    this.selenium = new DefaultSelenium("localhost", 4444, "*safari",
        "http://localhost:8080");
    this.selenium.start();
  }

  @Test public void testSequence() throws Exception {
    selenium.open("/sample-web/");
    selenium.type("index", "10");
    selenium.click("//input[@value='Calculate']");
    selenium.waitForPageToLoad("30000");
    assert selenium.isTextPresent("55");
  }

  @AfterClass(alwaysRun = true)
  public void stopSelenium() {
    this.selenium.stop();
  }
```

```
}
```

The POM to configure the sample-web-it is shown below.

```
<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/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.sonatype.mcookbook</groupId>
  <artifactId>sample-web-it</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>sample-web-it</name>
  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <configuration>
          <source>1.5</source>
          <target>1.5</target>
        </configuration>
      </plugin>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-dependency-plugin</artifactId>
        <executions>
          <execution>
            <id>copy</id>❶
            <phase>package</phase>
            <goals>
              <goal>copy</goal>
            </goals>
            <configuration>
              <artifactItems>
                <artifactItem>
                  <groupId>org.sonatype.mcookbook</groupId>
                  <artifactId>sample-web</artifactId>
                  <version>1.0-SNAPSHOT</version>
                  <type>war</type>
                  <overWrite>true</overWrite>
                  <destFileName>sample-web.war</destFileName>
                </artifactItem>
              </artifactItems>
              <outputDirectory>
                ${project.build.directory}/war
              </outputDirectory>
              <overWriteReleases>true</overWriteReleases>
              <overWriteSnapshots>true</overWriteSnapshots>
            </configuration>
          </execution>
        </executions>
      </plugin>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-surefire-plugin</artifactId>
        <configuration>
          <skip>true</skip>
        </configuration>
        <executions>
          <execution>
            <phase>integration-test</phase>❷
            <goals>
              <goal>test</goal>
            </goals>
```

```
          <configuration>
            <skip>false</skip>
          </configuration>
        </execution>
      </executions>
    </plugin>
    <plugin>
      <groupId>org.codehaus.mojo</groupId>
      <artifactId>selenium-maven-plugin</artifactId>
      <executions>
        <execution>
          <phase>pre-integration-test</phase>❸
          <goals>
            <goal>start-server</goal>
          </goals>
          <configuration>
            <background>true</background>
          </configuration>
        </execution>
      </executions>
    </plugin>
    <plugin>
      <groupId>org.mortbay.jetty</groupId>
      <artifactId>maven-jetty-plugin</artifactId>
      <version>6.1.22</version>
      <executions>
        <execution>
          <id>start-jetty</id>❹
          <phase>pre-integration-test</phase>
          <goals>
            <goal>run-war</goal>
          </goals>
          <configuration>
            <contextPath>sample-web</contextPath>
            <daemon>true</daemon>
            <webApp>
              ${project.build.directory}/war/sample-web.war
            </webApp>
          </configuration>
        </execution>
        <execution>
          <id>stop-jetty</id>❺
          <phase>post-integration-test</phase>
          <goals>
            <goal>stop</goal>
          </goals>
        </execution>
      </executions>
      <configuration>❻
        <stopPort>9991</stopPort>
        <stopKey>test</stopKey>
      </configuration>
    </plugin>
  </plugins>
</build>
<dependencies>
  <dependency>
    <groupId>org.testng</groupId>
    <artifactId>testng</artifactId>
    <version>5.10</version>
    <classifier>jdk15</classifier>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>org.seleniumhq.selenium.client-drivers</groupId>
    <artifactId>selenium-java-client-driver</artifactId>
    <version>1.0.1</version>
```

```
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>org.sonatype.mcookbook</groupId>❼
      <artifactId>sample-web</artifactId>
      <version>1.0-SNAPSHOT</version>
      <type>war</type>
    </dependency>
  </dependencies>
</project>
```

This POM contains the following configuration:

❶ The Maven Dependency plugin is configured to copy the sample-web web application archive to the ${project.build.directory}/war/sample-web.war file. The copy goal is bound to the package phase.

❷ The Surefire plugin is configured to skip all tests during the test phase and to execute all tests during the integration-test phase.

❸ The Selenium server is started during the pre-integration-test phase as a background process. This makes the Selenium server available to any integration tests that need to interact with a browser.

❹ The start-jetty execution starts the Jetty server in the pre-integration-test phase. This execution references the WAR downloaded by the Maven Dependency plugin and it also sets the context path to "sample-web". Setting daemon to "true" runs Jetty in the background and continues to progress through the Maven lifecycle. If daemon were set to "false", the Maven build would stop and wait for the Jetty process to complete.

❺ The stop-jetty execution stops the Jetty server after the integration tests have been executed.

❻ To stop the Jetty server we need to supply a stop port and a stop key. Both the run-war and stop goals of the Jetty plugin will use this configuration so it is defined at the plugin level instead of the execution level. The stop port defines the port number that Jetty will listen to for a stop command, and the stop key is a string that will be used by the jetty:stop goal to stop the Jetty process.

❼ Adding a dependency to the web application project that is being tested will make sure that this project is ordered after the web application in a multi-module build.

When you run the integration-test phase of this build, you will see that Maven runs through the lifecycle and starts Selenium and Jetty before running the integration tests.

```
$ mvn clean install
[INFO] Scanning for projects...
[INFO] ------------------------------------------------------------------------
[INFO] Building sample-web-it
[INFO]    task-segment: [clean, integration-test]
[INFO] ------------------------------------------------------------------------
[INFO] [clean:clean {execution: default-clean}]
...
[INFO] [resources:resources {execution: default-resources}]
...
[INFO] [resources:testResources {execution: default-testResources}]
...
[INFO] [compiler:testCompile {execution: default-testCompile}]
[INFO] Compiling 1 source file to
~/Code/sonatype/maven-cookbook/mcookbook-examples/integrate/sample-web-it/
target/test-classes
[INFO] [surefire:test {execution: default-test}]
[INFO] Tests are skipped.
[INFO] [jar:jar {execution: default-jar}]
...
[INFO] [dependency:copy {execution: copy}]
...
[INFO] [selenium:start-server {execution: default}]
Created dir: ~/maven-cookbook/mcookbook-examples/integrate/sample-web-it/
target/selenium
Launching Selenium Server
Waiting for Selenium Server...
...
[INFO] [dependency:copy {execution: copy}]
[INFO] Configured Artifact: org.sonatype.mcookbook:sample-web:1.0-SNAPSHOT:war
[INFO] Copying sample-web-1.0-SNAPSHOT.war to
```

```
~/maven-cookbook/mcookbook-examples/integrate/sample-web-it/
target/war/sample-web.war
[INFO] [jetty:run-war {execution: start-jetty}]
[INFO] Configuring Jetty for project: sample-web-it
2009-11-29 04:21:37.994:INFO::Logging to STDERR via org.mortbay.log.StdErrLog
[INFO] Context path = /sample-web
[INFO] Starting jetty 6.1.22 ...
2009-11-29 04:21:38.683:INFO::Started SelectChannelConnector@0.0.0.0:8080
[INFO] Started Jetty Server
[INFO] [surefire:test {execution: default}]

-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running TestSuite
04:21:39,436 INFO  [org.mortbay.util.Credential] Checking Resource aliases
...
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 3.597 sec

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

[INFO] [jetty:stop {execution: stop-jetty}]
[INFO] Stopping server 0
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 13 seconds
[INFO] Finished at: Sun Nov 29 04:21:42 CST 2009
[INFO] Final Memory: 48M/99M
[INFO] ------------------------------------------------------------------------
2009-11-29 04:21:43.201:INFO::Shutdown hook executing
2009-11-29 04:21:43.822:INFO::Shutdown hook complete
```

Chapter 9. Releasing Software with Maven

9.1. Introduction

In Maven existing four different possibilities to release an artifact. The release of an artifact has different meaning in different areas where you release an artifact.

9.2. Install Artifact

Installing an artifact means to deploy an artifact into the users Maven repository (`~/.m2/repository`). This makes it possible to use an artifact as a dependency for an other project by this user on the same machine.

9.3. Deploying Artifact

Deploying is meant to be releasing an artifact (SNAPSHOT) into a repository outside users machine which means to make it usable by others of the team into their projects.

9.4. Releasing an Artifact

Releasing an artifact in Maven comprises of two separated steps which are the **`release:prepare`** and the **`release:perform`**. The two steps can be manually or automatically by any Continuous Integration system.

9.4.1. Release Prepare

If you think your project reached a particular step of your development the time comes to do a release of your project which simply can be achieved by using Mavens release process. The first step is to do **`mvn release:prepare`** in your current state of your project. The **`release:prepare`** will check if you missed code changes to check in and take a look at the projects dependencies that it has no SNAPSHOT dependencies and furthermore change the versions in the POMs from X-SNAPSHOT to a new version (you will be prompted for that). The next thing is that the SCM information in the projects POM will be changed to point to the final destination of the tag and all tests will be run against the modified POMs to confirm everything is working fine. Now the modified POMs will be committed and the code in the SCM will be tagged by a version name (prompted for) and change the POMs to use a new version y-SNAPSHOT (these values have been prompted for before) and finally commit the modified POMs into the SCM system.

XXXXX

9.4.2. Release Perform

The **`release:perform`** will checkout the code from the previously create tag and run the predefined Maven goals (by default: **`deploy site-deploy`**) on it.

Chapter 10. Repository Management

10.1. Installing and Running Nexus Open Source

10.1.1. Task

You need to download and install Nexus Open Source to proxy and host Maven repositories.

10.1.2. Action

To download and install Sonatype Nexus Open Source, follow the instructions in Section 2.1.1, "Downloading Nexus Open Source",[1] and Section 2.2.2, "Installing Nexus Open Source",[2] from "Repository Management with Nexus".

If you've installed Nexus on the default port 8081 listening on localhost, going to http://localhost:8081/nexus/ will load the interface shown in Figure 10.3, “Crowd Menu Link under the Security Section of the Nexus Menu”.

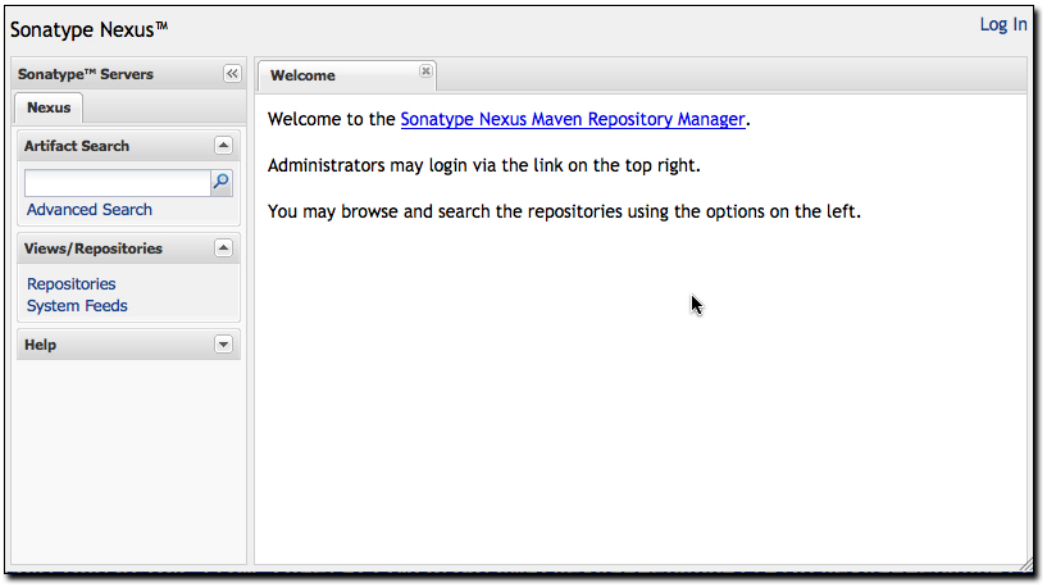

Figure 10.1. Nexus Open Source Prior to Authentication

Clicking on the Log In link in the upper right-hand corner of the window, will prompt you for a username and password in the Nexus Log In dialogin shown in Figure 10.2, “The Nexus Log In Dialog”. The default administrative username is "admin" and the default administrative password is "admin123".

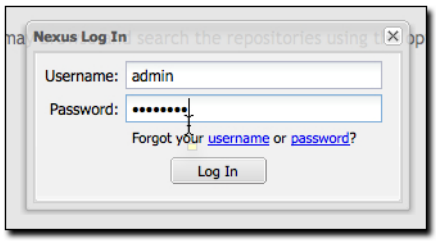

Figure 10.2. The Nexus Log In Dialog

[1] http://www.sonatype.com/books/nexus-book-stage/reference/install.html#installing-sect-dl-open
[2] http://www.sonatype.com/books/nexus-book-stage/reference/ch02s02.html#installing-sect-open-source

Once you have successfully authenticated as an administrative user, you will see the full Nexus user interface as shown in Figure 10.3, “Crowd Menu Link under the Security Section of the Nexus Menu”.

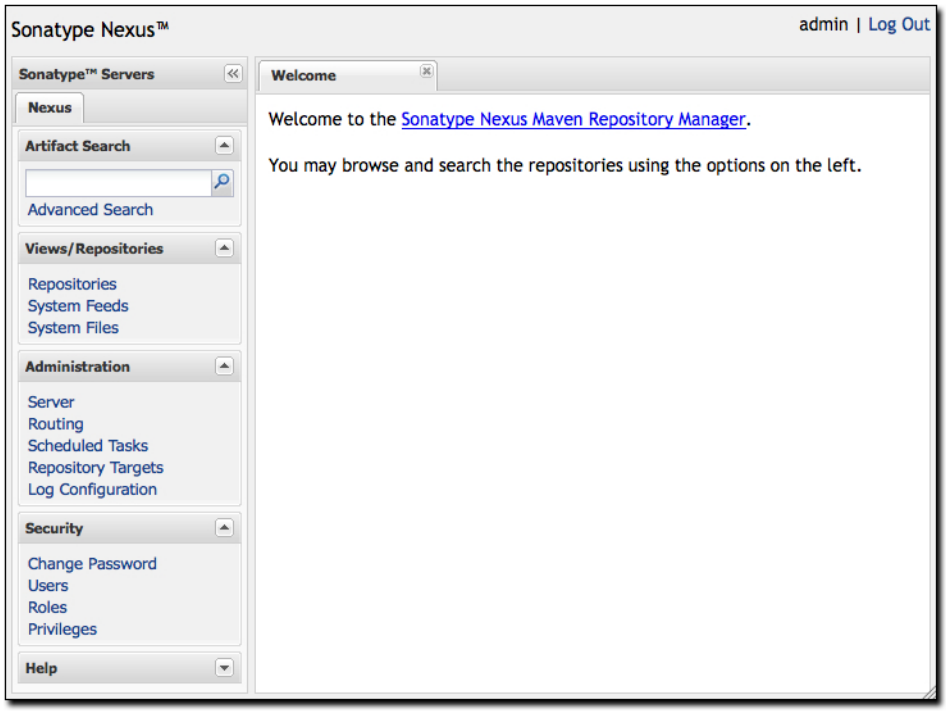

Figure 10.3. Crowd Menu Link under the Security Section of the Nexus Menu

10.. Proxying a Remote Repository

10..1. Task

You and your team or organization are heavy users of remote repositories. Every time a user installs a development environment, they end up downloading data directly from the remote repository. You want to create a caching proxy of the remote repository that will increase the speed of your builds and give you some stability in case the remote repository is unavailable.

10..2. Action

To configure a new Proxy repository, open the Repositories panel by clicking on Repositories under Views/Repositories section of the Nexus menu. Clicking Repositories should display the Repositories panel shown in Figure 10.4, “Opening the Repositories Panel”.

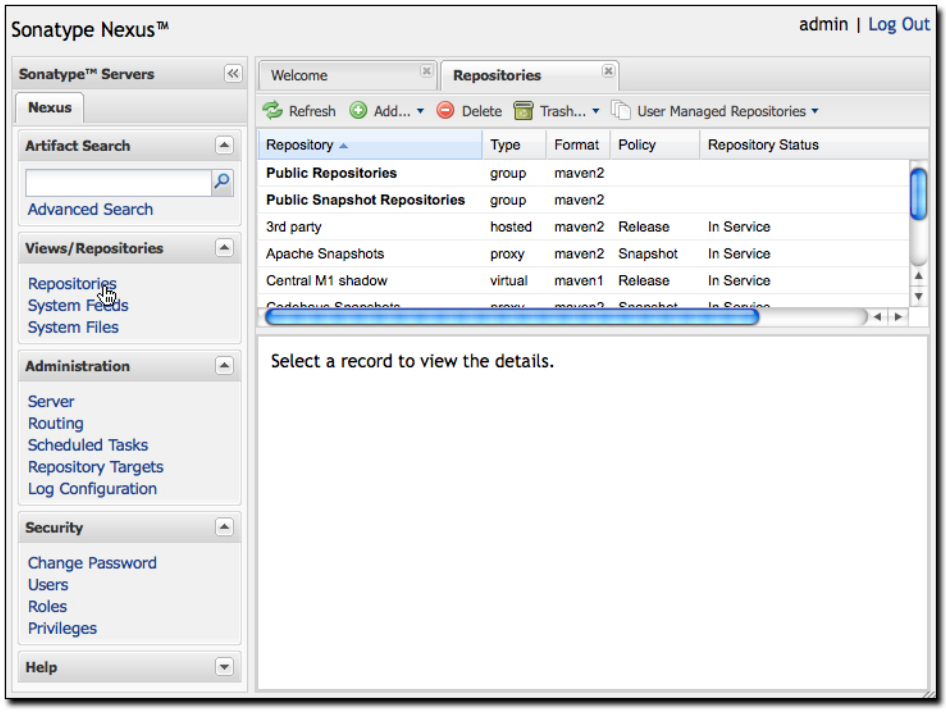

Figure 10.4. Opening the Repositories Panel

To create a new proxy repository, click on the Add... button and select Proxy Repository from the dropdown as shown in Figure 10.5, “Creating a New Proxy Repository”.

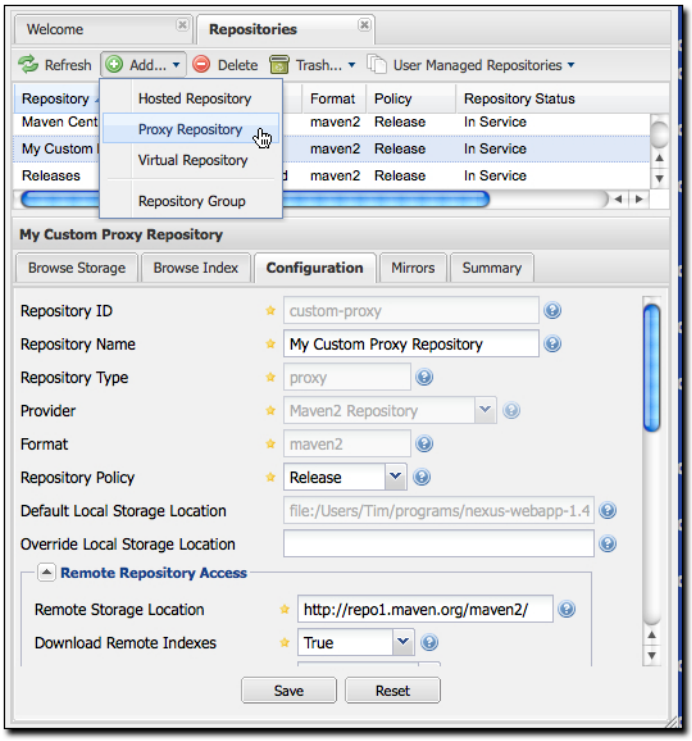

Figure 10.5. Creating a New Proxy Repository

When you create a new proxy repository, you'll need to choose a Repository ID. A Repository ID is an internal identifier which is used in the URL that you will connect your clients to. The standard for Repository IDs is to use a lower case word or words separated by a hyphen. Examples of valid Repository IDs are: "central", "internal", and "custom-snapshots".

Other important fields in the form shown in Figure 10.5, “Creating a New Proxy Repository”, are: Repository Policy, Remote Storage Location, and Download Remote Indexes. The repository policy controls what artifacts can be stored in a repository: snapshot artifacts or release artifacts. The Remote Storage Location is the URL of the remote repository, and Download Remote Indexes tells Nexus to download an index that will enable searching and browsing from the remote repository.

10.3. Browsing a Nexus Repository

10.3.1. Task

You want to browse the contents of a Nexus repository.

10.3.2. Action

If your Nexus repository is a proxy repository, you will have two browsing tabs available: Browse Storage and Browse Index. If your Nexus repository is a hosted repository, you will only see the Browse Storage option, and if you are browsing a repository group, you will see both Browse Storage and Browse Index.

The Browse Storage tab shown in Figure 10.6, “Browsing Repository Storage”, allows you to see the artifacts that available available in the local disk storage associated with a repostiory. If you are browsing a hosted repository, the storage contains the entire repository. If you are browsing a proxy repository, the Browse Storage tab will display only the artifacts that have been referenced, downloaded, and cached by Nexus.

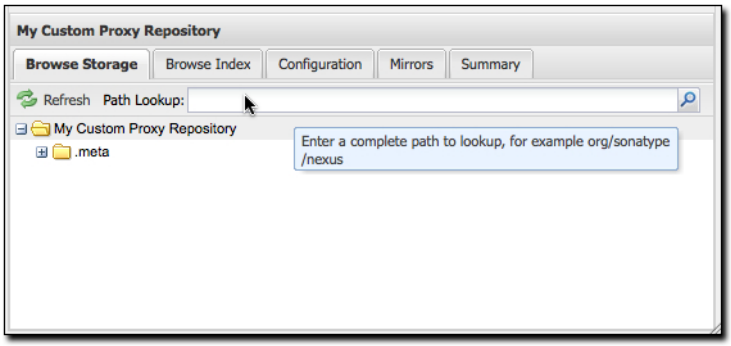

Figure 10.6. Browsing Repository Storage

If you are browsing a proxy repository and you have configured Nexus to download the remote repository index, you will be able to browse the entire repository in the Browse Index tab as shown in Figure 10.7, “Crowd Menu Link under the Security Section of the Nexus Menu”.

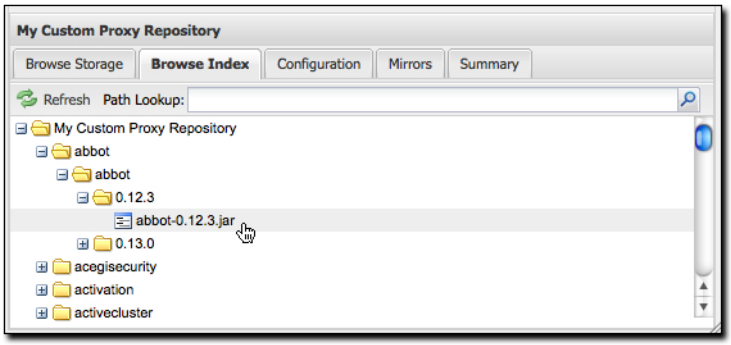

Figure 10.7. Crowd Menu Link under the Security Section of the Nexus Menu

10.4. Configuring a Nexus Repository

10.4.1. Task

You need to change the configuration of a Nexus repository.

10.4.2. Action

To configure a Nexus repository, click on Repositories under the Views/Repositories section of the Nexus menu, and then select the repository you wish to configure. Once the repository is select, click on the Configuration tab to display the form shown in Figure 10.8, “Crowd Menu Link under the Security Section of the Nexus Menu”.

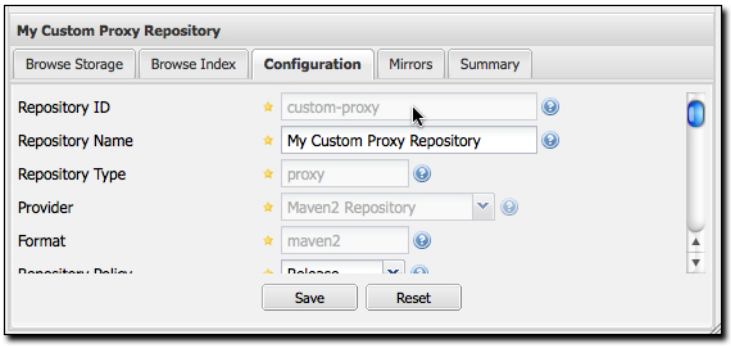

Figure 10.8. Crowd Menu Link under the Security Section of the Nexus Menu

10.5. Viewing Summary Information for Nexus Repositories

10.5.1. Task

You need to view some summary information about a particular repository in Nexus.

10.5.2. Action

To view summary information about a Nexus repository, click on Repositories under the Views/Repositories section of the Nexus menu, and then select the repository you wish to view. Once the repository is select, click on the Summary tab to display the form shown in Figure 10.9, “Crowd Menu Link under the Security Section of the Nexus Menu”.

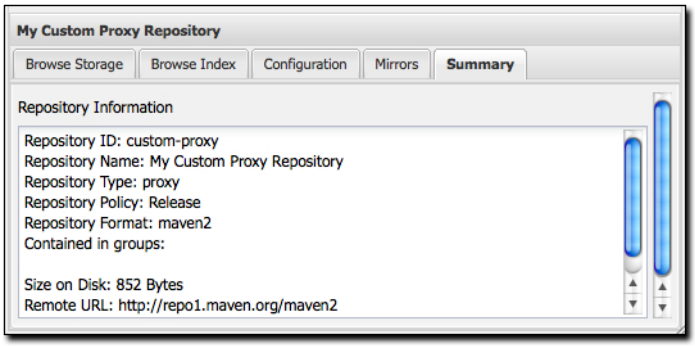

Figure 10.9. Crowd Menu Link under the Security Section of the Nexus Menu

10.6. Using Mirrors for Proxy Repositories

10.6.1. Task

You want to help ease the traffic and bandwidth burden on the Central Maven repository by configuring your Nexus instance to use a mirror to download artifacts.

10.6.2. Action

Once you have created a proxy repository, you should configure Nexus to download artifacts from one or more mirrors. To configure mirrors, click on Repositories under the View/Repositories section of the Nexus menu, select the proxy repository you want to configure, and then select the Mirrors tab for this repository. Clicking on the Mirrors tab will display the form shown in Figure 10.10, “Configuring Mirrors for Proxy Repositories”.

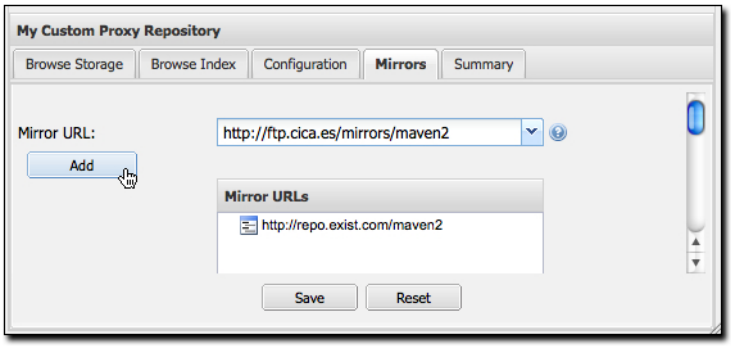

Figure 10.10. Configuring Mirrors for Proxy Repositories

The Central Maven repository contains an XML file which will be used to populate the mirrors dropdown shown in Figure 10.10, “Configuring Mirrors for Proxy Repositories”. To configure a mirror, select it from the dropdown, and click on the Add button to add the mirror to the list of Mirror URLs.

10.6.3. Details

When a proxy repository is confgured to retrieve artifacts from a Mirror, it will consult each configured mirror in the order shown in the Mirror URLs list from Figure 10.10, “Configuring Mirrors for Proxy Repositories”. While Nexus will retrieve POM information, checksums, and PGP signatures from the original repository, it will retrieve all artfiacts directly from the first available mirror.

10.. Grouping Repositories

10..1. Task

You want to consolidate access to multiple repositories with a single repository URL.

10..2. Action

Nexus Open Source provides for the ability to group one or more repositories into a Repository Group. The default installation of Nexus is preconfigured with two Repository Groups: Public Repositories and Public Snapshot Repositories. Public Repositories groups all of the repositories with a Release policy together, and Public Snapshot Repositories groups all of the repositories with a Snapshot policy together. To add a repository to the Public Repositories group, click on Repositories under the View/Repositories section of the Nexus menu, and then click on the Repository Group that you want to add a Repository to. Once the group is selected, click on the Configuration tab shown in Figure 10.11, “Adding a Repository to a Repository Group”, and drag the repository you wich to add from the Available Repositories list to the Ordered Group Repositories list.

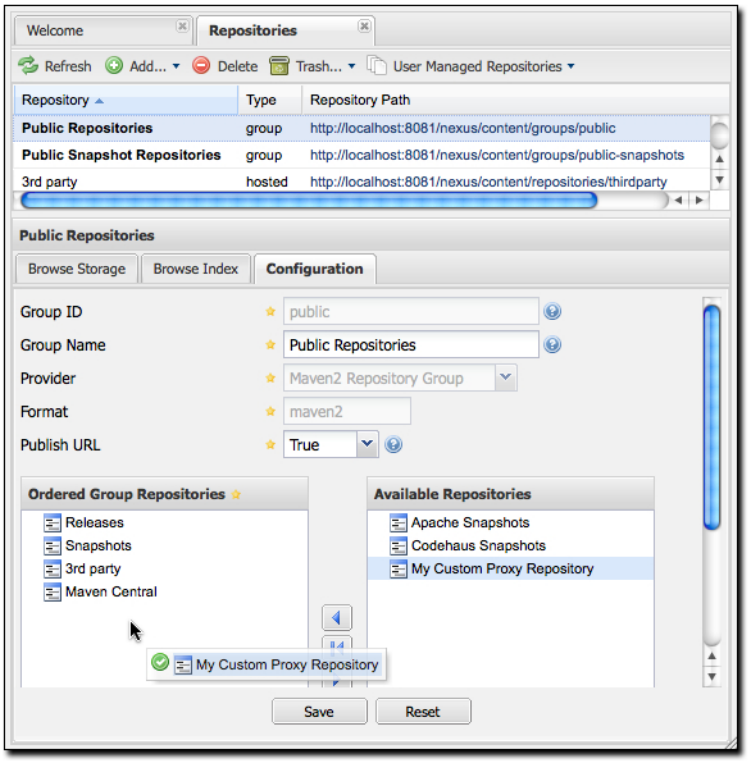

Figure 10.11. Adding a Repository to a Repository Group

To add a new Repository Group, click on the Add.. button and choose Repository Group.

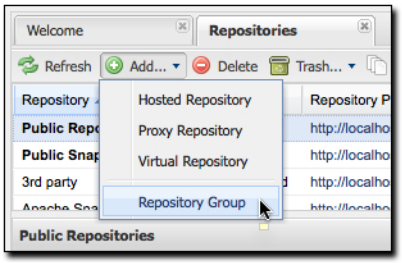

Figure 10.12. Adding a New Repository Group

10.8. Installing Nexus Professional

10.8.1. Task

You need to download and install Nexus Professional to integrate your repository manager with enteprise security, manage the procurement of software artifacts, and track staged software releases.

10.8.2. Action

To download and install Sonatype Nexus Professional, follow the instructions in Section 2.1.2, "Downloading Nexus Professional",[3] and Section 2.2.3, "Installing Nexus Professional",[4] from "Repository Management with Nexus".

[3] http://www.sonatype.com/books/nexus-book-stage/reference/install.html#install-sect-download
[4] http://www.sonatype.com/books/nexus-book-stage/reference/ch02s02.html#install-sect-pro

10.9. Configuring a Staging Repository for Deployment in Nexus Professional

There might be any number of reasons why you might want to require your developers to use a staging repository. You might want to make sure that software artifacts contain valid POMs using a Staging Ruleset, or you might want to make sure that your process guarantees that QA has the final decision making authority to either promote or drop a release candidate. Staged software releases are very easy to implement using the Staging Plugin in Nexus Professional[5]. If you want to learn how to make a staged release, you can watch this video[6]. or read the remainder of this section.

10.9.1. Task

Deploy an artifact to a staging repository in Nexus Professional.

10.9.2. Action

First step, is to download Nexus Professional if you don't already have it installed, you can do so, by going to the Nexus Professional product page and clicking on the download link in the right-hand menu. Once you've downloaded Nexus, log in as an administrator as you must have administrative rights to perform the following configuration:

1. From Nexus, click on the 'Repositories' link in left navigation.
2. Click on the repository you want to use, or create one (If you need help with that take a look at the Nexus book, here[7])
3. Click on the 'Configuration' tab.
4. Set the 'Deployment Policy' field to "Read Only".
5. Click the Save button.

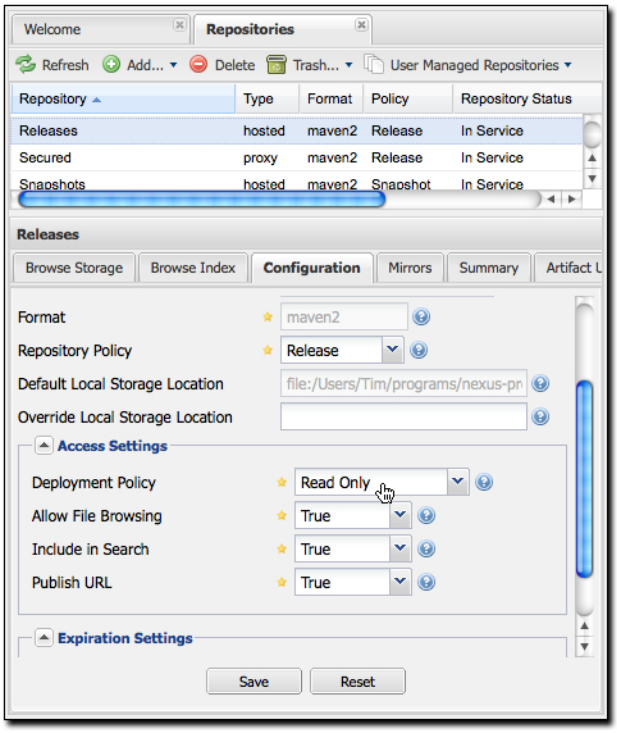

Figure 10.13. Prohibit Direct Deployment to a Release Repository

[5] http://www.sonatype.com/products/nexus
[6] http://www.sonatype.com/people/2009/01/nexus-professional-what-is-staging/

That is it! Users will not be able to deploy or upload artifacts directly to the repository. All artifacts must be staged and promoted to this repository. For more information on staging and promoting take a look at this[8], or with the maven plugin, here[9].

Next, create a staging repository. To create a staging repository:

1. From Nexus, click on the 'Staging' link in left navigation.
2. Click the 'Add' button then the 'Staging Profile' item.
3. Enter the following information:
   - Profile Name: Staging Demo Profile
   - Profile Repository Target: All (Maven2)
   - Staging Repository ID Template: staging-demo
   - Staging Repository Name Template: Staging Demo
   - Staging Repository Template: Default Release Hosted Repository Template
   - Target Groups: Public Releases
4. Click the 'Save' button

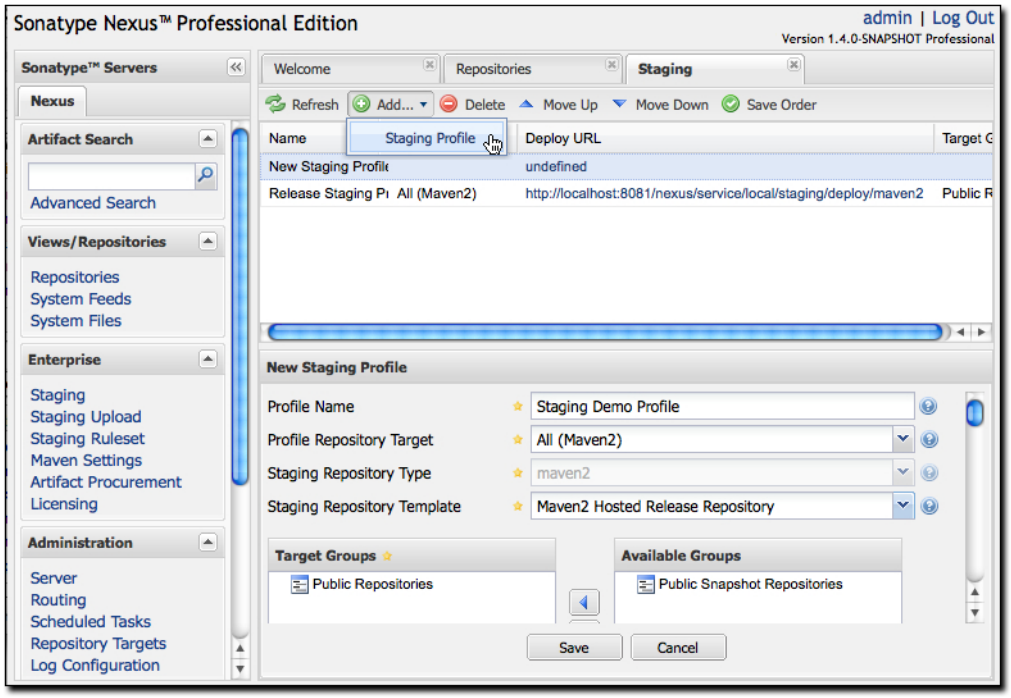

Figure 10.14. Configuring a Staging Profile

Next, assign new permissions to users who will need to deploy to the staging repository. A new role is created for each staging profile that is created (in this example the role is 'Staging Deployer: (Staging Demo Profile)'). Assign the new role to your users. You can find more details on user management, here[10].

[8] http://www.sonatype.com/books/nexus-book/reference/staging-sect-managing-staging.html
[9] http://www.sonatype.com/books/nexus-book/reference/ch09s07.html
[10] http://www.sonatype.com/books/nexus-book/reference/config.html#config-sect-managing-users

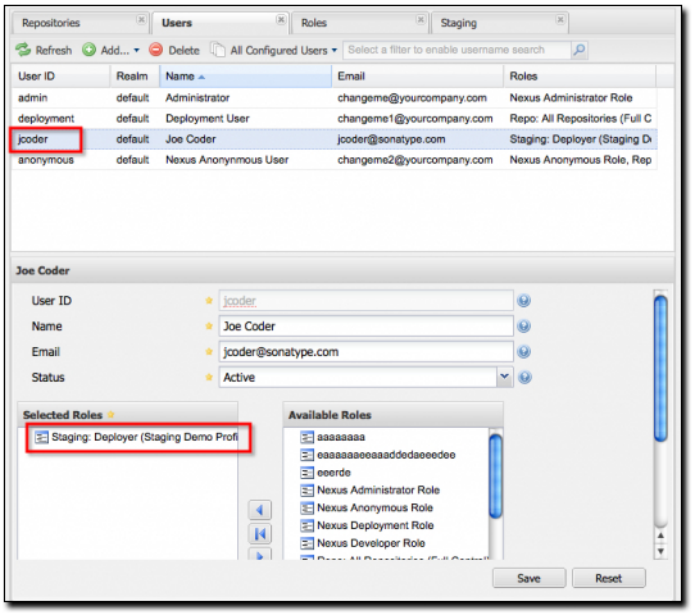

Figure 10.15. Assigning Staging Roles

Appendix A. Creative Commons License

This book is licensed under a Creative Commons Attribution-Noncommercial-No Derivative Works 3.0 United States license. For more information about this license, see http://creativecommons.org/licenses/by-nc-nd/3.0/us/. You are free to share, copy, distribute, display, and perform the work under the following conditions:

- You must attribute the work to Sonatype, Inc. with a link to http://www.sonatype.com.
- You may not use this work for commercial purposes.
- You may not alter, transform, or build upon this work.

If you redistribute this work on a web page, you must include the following link with the URL in the about attribute listed on a single line (remove the backslashes and join all URL parameters):

```
<div xmlns:cc="http://creativecommons.org/ns#"
     about="http://creativecommons.org/license/results-one?q_1=2&q_1=1\
            &field_commercial=n&field_derivatives=n&field_jurisdiction=us\
            &field_format=StillImage&field_worktitle=Maven%3A+\Guide\
            &field_attribute_to_name=Sonatype%2C+Inc.\
            &field_attribute_to_url=http%3A%2F%2Fwww.sonatype.com\
            &field_sourceurl=http%3A%2F%2Fwww.sonatype.com%2Fbook\
            &lang=en_US&language=en_US&n_questions=3">
   <a rel="cc:attributionURL" property="cc:attributionName"
      href="http://www.sonatype.com">Sonatype, Inc.</a> /
   <a rel="license"
        href="http://creativecommons.org/licenses/by-nc-nd/3.0/us/">
     CC BY-NC-ND 3.0</a>
</div>
```

When downloaded or distributed in a jurisdiction other than the United States of America, this work shall be covered by the appropriate ported version of Creative Commons Attribution-Noncommercial-No Derivative Works 3.0 license for the specific jurisdiction. If the Creative Commons Attribution-Noncommercial-No Derivative Works version 3.0 license is not available for a specific jurisdiction, this work shall be covered under the Creative Commons Attribution-Noncommercial-No Derivate Works version 2.5 license for the jurisdiction in which the work was downloaded or distributed. A comprehensive list of jurisdictions for which a Creative Commons license is available can be found on the Creative Commons International web site at http://creativecommons.org/international.

If no ported version of the Creative Commons license exists for a particular jurisdiction, this work shall be covered by the generic, unported Creative Commons Attribution-Noncommercial-No Derivative Works version 3.0 license available from http://creativecommons.org/licenses/by-nc-nd/3.0/.

A.1. Creative Commons BY-NC-ND 3.0 US License

Creative Commons Attribution-NonCommercial-NoDerivs 3.0 United States[1]

THE WORK (AS DEFINED BELOW) IS PROVIDED UNDER THE TERMS OF THIS CREATIVE COMMONS PUBLIC LICENSE ("CCPL" OR "LICENSE"). THE WORK IS PROTECTED BY COPYRIGHT AND/OR OTHER APPLICABLE LAW. ANY USE OF THE WORK OTHER THAN AS AUTHORIZED UNDER THIS LICENSE OR COPYRIGHT LAW IS PROHIBITED.

BY EXERCISING ANY RIGHTS TO THE WORK PROVIDED HERE, YOU ACCEPT AND AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE. TO THE EXTENT THIS LICENSE MAY BE CONSIDERED TO BE A CONTRACT, THE LICENSOR GRANTS YOU THE RIGHTS CONTAINED HERE IN CONSIDERATION OF YOUR ACCEPTANCE OF SUCH TERMS AND CONDITIONS.

[1] http://creativecommons.org/licenses/by-nc-nd/3.0/us/legalcode

1. Definitions

    a. "Collective Work" means a work, such as a periodical issue, anthology or encyclopedia, in which the Work in its entirety in unmodified form, along with one or more other contributions, constituting separate and independent works in themselves, are assembled into a collective whole. A work that constitutes a Collective Work will not be considered a Derivative Work (as defined below) for the purposes of this License.

    b. "Derivative Work" means a work based upon the Work or upon the Work and other pre-existing works, such as a translation, musical arrangement, dramatization, fictionalization, motion picture version, sound recording, art reproduction, abridgment, condensation, or any other form in which the Work may be recast, transformed, or adapted, except that a work that constitutes a Collective Work will not be considered a Derivative Work for the purpose of this License. For the avoidance of doubt, where the Work is a musical composition or sound recording, the synchronization of the Work in timed-relation with a moving image ("synching") will be considered a Derivative Work for the purpose of this License.

    c. "Licensor" means the individual, individuals, entity or entities that offers the Work under the terms of this License.

    d. "Original Author" means the individual, individuals, entity or entities who created the Work.

    e. "Work" means the copyrightable work of authorship offered under the terms of this License.

    f. "You" means an individual or entity exercising rights under this License who has not previously violated the terms of this License with respect to the Work, or who has received express permission from the Licensor to exercise rights under this License despite a previous violation.

2. Fair Use Rights. Nothing in this license is intended to reduce, limit, or restrict any rights arising from fair use, first sale or other limitations on the exclusive rights of the copyright owner under copyright law or other applicable laws.

3. License Grant. Subject to the terms and conditions of this License, Licensor hereby grants You a worldwide, royalty-free, non-exclusive, perpetual (for the duration of the applicable copyright) license to exercise the rights in the Work as stated below:

    a. to reproduce the Work, to incorporate the Work into one or more Collective Works, and to reproduce the Work as incorporated in the Collective Works; and,

    b. to distribute copies or phonorecords of, display publicly, perform publicly, and perform publicly by means of a digital audio transmission the Work including as incorporated in Collective Works.

    The above rights may be exercised in all media and formats whether now known or hereafter devised. The above rights include the right to make such modifications as are technically necessary to exercise the rights in other media and formats, but otherwise you have no rights to make Derivative Works. All rights not expressly granted by Licensor are hereby reserved, including but not limited to the rights set forth in Sections 4(d) and 4(e).

4. Restrictions.The license granted in Section 3 above is expressly made subject to and limited by the following restrictions:

    a. You may distribute, publicly display, publicly perform, or publicly digitally perform the Work only under the terms of this License, and You must include a copy of, or the Uniform Resource Identifier for, this License with every copy or phonorecord of the Work You distribute, publicly display, publicly perform, or publicly digitally perform. You may not offer or impose any terms on the Work that restrict the terms of this License or the ability of a recipient of the Work to exercise the rights granted to that recipient under the terms of the License. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties. When You distribute, publicly display, publicly perform, or publicly digitally perform the Work, You may not impose any technological measures on the Work that restrict the ability of a recipient of the Work from You to exercise the rights granted to that recipient under the terms of the License. This Section 4(a) applies to the Work as incorporated in a Collective Work, but this does not require the Collective Work apart from the Work itself to be made subject to the terms of this License. If You create a Collective Work, upon notice from any Licensor You must, to the extent practicable, remove from the Collective Work any credit as required by Section 4(c), as requested.

    b. You may not exercise any of the rights granted to You in Section 3 above in any manner that is primarily intended for or directed toward commercial advantage or private monetary compensation. The exchange of the Work for other copyrighted

works by means of digital file-sharing or otherwise shall not be considered to be intended for or directed toward commercial advantage or private monetary compensation, provided there is no payment of any monetary compensation in connection with the exchange of copyrighted works.

c. If You distribute, publicly display, publicly perform, or publicly digitally perform the Work (as defined in Section 1 above) or Collective Works (as defined in Section 1 above), You must, unless a request has been made pursuant to Section 4(a), keep intact all copyright notices for the Work and provide, reasonable to the medium or means You are utilizing: (i) the name of the Original Author (or pseudonym, if applicable) if supplied, and/or (ii) if the Original Author and/or Licensor designate another party or parties (e.g. a sponsor institute, publishing entity, journal) for attribution ("Attribution Parties") in Licensor's copyright notice, terms of service or by other reasonable means, the name of such party or parties; the title of the Work if supplied; to the extent reasonably practicable, the Uniform Resource Identifier, if any, that Licensor specifies to be associated with the Work, unless such URI does not refer to the copyright notice or licensing information for the Work. The credit required by this Section 4(c) may be implemented in any reasonable manner; provided, however, that in the case of a Collective Work, at a minimum such credit will appear, if a credit for all contributing authors of the Collective Work appears, then as part of these credits and in a manner at least as prominent as the credits for the other contributing authors. For the avoidance of doubt, You may only use the credit required by this clause for the purpose of attribution in the manner set out above and, by exercising Your rights under this License, You may not implicitly or explicitly assert or imply any connection with, sponsorship or endorsement by the Original Author, Licensor and/or Attribution Parties, as appropriate, of You or Your use of the Work, without the separate, express prior written permission of the Original Author, Licensor and/or Attribution Parties.

d. For the avoidance of doubt, where the Work is a musical composition:

   i. Performance Royalties Under Blanket Licenses. Licensor reserves the exclusive right to collect whether individually or, in the event that Licensor is a member of a performance rights society (e.g. ASCAP, BMI, SESAC), via that society, royalties for the public performance or public digital performance (e.g. webcast) of the Work if that performance is primarily intended for or directed toward commercial advantage or private monetary compensation.

   ii. Mechanical Rights and Statutory Royalties. Licensor reserves the exclusive right to collect, whether individually or via a music rights agency or designated agent (e.g. Harry Fox Agency), royalties for any phonorecord You create from the Work ("cover version") and distribute, subject to the compulsory license created by 17 USC Section 115 of the US Copyright Act (or the equivalent in other jurisdictions), if Your distribution of such cover version is primarily intended for or directed toward commercial advantage or private monetary compensation.

e. Webcasting Rights and Statutory Royalties. For the avoidance of doubt, where the Work is a sound recording, Licensor reserves the exclusive right to collect, whether individually or via a performance-rights society (e.g. SoundExchange), royalties for the public digital performance (e.g. webcast) of the Work, subject to the compulsory license created by 17 USC Section 114 of the US Copyright Act (or the equivalent in other jurisdictions), if Your public digital performance is primarily intended for or directed toward commercial advantage or private monetary compensation.

5. Representations, Warranties and Disclaimer

   UNLESS OTHERWISE MUTUALLY AGREED TO BY THE PARTIES IN WRITING, LICENSOR OFFERS THE WORK AS-IS AND ONLY TO THE EXTENT OF ANY RIGHTS HELD IN THE LICENSED WORK BY THE LICENSOR. THE LICENSOR MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MARKETABILITY, MERCHANTIBILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.

6. Limitation on Liability. EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. Termination

    a. This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Collective Works (as defined in Section 1 above) from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.

    b. Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

8. Miscellaneous

    a. Each time You distribute or publicly digitally perform the Work (as defined in Section 1 above) or a Collective Work (as defined in Section 1 above), the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.

    b. If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

    c. No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.

    d. This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements or representations with respect to the Work not specified here. Licensor shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.

Creative Commons Notice

Creative Commons is not a party to this License, and makes no warranty whatsoever in connection with the Work. Creative Commons will not be liable to You or any party on any legal theory for any damages whatsoever, including without limitation any general, special, incidental or consequential damages arising in connection to this license. Notwithstanding the foregoing two (2) sentences, if Creative Commons has expressly identified itself as the Licensor hereunder, it shall have all rights and obligations of Licensor.

Except for the limited purpose of indicating to the public that the Work is licensed under the CCPL, Creative Commons does not authorize the use by either party of the trademark "Creative Commons" or any related trademark or logo of Creative Commons without the prior written consent of Creative Commons. Any permitted use will be in compliance with Creative Commons' then-current trademark usage guidelines, as may be published on its website or otherwise made available upon request from time to time. For the avoidance of doubt, this trademark restriction does not form part of this License.

Creative Commons may be contacted at http://creativecommons.org/.

Appendix B. Book Revision History

Many readers have been asking us to keep track of specific changes to the book content, the following sections list changes made to the book in reverse chronological order starting with 0.4.

B.1. Changes in Edition 0.3

The following changes were introduced in Edition 0.3:

- Update the book title to "The Maven Cookbook". (MVNCOOK-74[1])
- Removed versions from the cover of the PDF. (MVNCOOK-72[2])
- Updated the business address on the Copyright page. (MVNCOOK-71[3])
- Fixed a trivial error in Section 1.3, “Starting an OSGi Container”. "not you want" changed to "now you want". (MVNCOOK-68[4])

B.2. Changes in Edition 0.2

The following changes were introduced in Edition 0.2:

- Reformatted book to 8.5" x 11" format.
- Fixed various line wrapping issues throughout the OSGi chapter.

B.3. Changes in Edition 0.1.4

The following changes were introduced in Edition 0.1.4:

- Added Chapter 10, Repository Management. (MVNCOOK-35[5])
  - Added a simple recipe that references the Nexus Book for Section 10.1, “Installing and Running Nexus Open Source”, and Section 10.8, “Installing Nexus Professional”. (MVNBOOK-37[6], MVNBOOK-38[7])
  - Added Section 10., “Proxying a Remote Repository”. (MVNBOOK-39[8])
  - Added Section 10.3, “Browsing a Nexus Repository”. (MVNCOOK-46[9])
  - Added Section 10.4, “Configuring a Nexus Repository”. (MVNCOOK-47[10])
  - Added Section 10.5, “Viewing Summary Information for Nexus Repositories”. (MVNCOOK-48[11])
  - Added Section 10.6, “Using Mirrors for Proxy Repositories”. (MVNCOOK-49[12])
  - Added Section 10., “Grouping Repositories”. (MVNCOOK-41[13])
  - Added Section 10.9, “Configuring a Staging Repository for Deployment in Nexus Professional”. (MVNCOOK-48[14])
- Updated the book cover. (MVNCOOK-36[15])
- Reduced the heading level displayed in the Table of Contents to Level 1. (MVNCOOK-43[16])
- Added a Table of Examples to the Book PDF. (MVNCOOK-45[17])
- Added a Table of Figures to the Book PDF. (MVNCOOK-44[18])

- Added Linux trademark information to Copyright. (MVNCOOK-31[19])
- Added Brian Demers to the list of authors. (MVNCOOK-49[20])

B.4. Changes in Edition 0.1.3

The following changes were introduced in Edition 0.1.3:

- Added a new Chapter on Ruby and Maven
- Added a new Chapter on Releasing Software with Maven
- Added a new Chatper on Unit Testing with Maven
- Added a new Chapter on Integration Testing with Maven
- Added a new Recipe for Creating an Ant Maven Plugin
- Added a new Recipe for Creating a Groovy Maven Plugin
- Prepared Book for Printing:
  - Book resized to Crown Quarto (7.444" x 9.681"). (MVNCOOK-18[21])
  - Assigned the Cookbook an ISBN. (MVNCOOK-19[22])
  - Created a Title Page for the Cookbook. (MVNCOOK-20[23])
  - Created a Copyright Page for the Cookbook. (MVNCOOK-21[24])
  - Embedded Fonts in the PDF. (MVNCOOK-22[25])
  - Standardized on 0.5" Margins. (MVNCOOK-23[26])
  - Added Text of Creative Commons License to Copyright. (MVNCOOK-26[27])
  - Automated the Generation of Print PDF figures. (MVNCOOK-24[28])
  - Started using roles for image objects in print output. (MVNCOOK-25[29])
- Changed the name of the book to Maven Cookbook. (MVNCOOK-27[30])
- Address feedback from proof of 0.1.2 PDF: (MVNCOOK-8[31])
  - Fixed a typo on Page 53. (MVNCOOK-9[32])
  - Fixed a package name error on Page 52. (MVNCOOK-10[33])
  - Fixed a grammar issue on Page 52. (MVNCOOK-11[34])
  - Fixed a spelling issues on Page 5. (MVNCOOK-12[35])
  - Removed duplicate the from Page 41. (MVNCOOK-13[36])
  - Fixed sentence structure on Page 40. (MVNCOOK-14[37])
  - Fixed a Spelling Problem on Page 33. (MVNCOOK-16[38])
  - Removed duplicate punctuation from Page 1. (MVNCOOK-17[39])

B.5. Changes in Edition 0.1.2

The following changes were introduced in Edition 0.1.2:

- Added Scribd API integration to the Maven Cookbook Maven build (yes, the book has a Maven build). This book now uses the Maven Scribd Plugin to publish the book to Scribd.

B.6. Changes in Edition 0.1.1

The following changes were introduced in Edition 0.1.1:

- Edited the OSGi Chapter based on Stuart's feedback, this chapter now contains more information about Equinox and Knopflerfish, it also adds some clarifications to the introduction.
- Added three simple chapters about scripting. Each of these chapters focuses on the same two recipes: how to include an inline script, and how to execute an external script. These three chapters are:
  - Groovy Maven
  - Ant and Maven
  - Scala and Maven

  The long-term plan for these chapter is to build out the content to replace the existing (now outdated) chapters from the Definitive Guide on Groovy and Ant.

B.7. Changes in Edition 0.1

The following changes were introduced in Edition 0.1:

- Seeded the initial version of the book with the source of the Nexus book.
- Create a cover for the Maven Cookbook.
- Write a Chapter on OSGi Development

Index