Red Hat Fuse 7.1

Installing on Apache Karaf

Installing Red Hat Fuse on the Apache Karaf container

Last Updated: 2018-09-25

Red Hat Fuse 7.1 Installing on Apache Karaf

Installing Red Hat Fuse on the Apache Karaf container

Legal Notice

Abstract

It is easy to install Red Hat Fuse and tailor the installation to a particular environment.

Table of Contents

CHAPTER 1. INSTALLATION PREREQUISITES

Before attempting to install and use Fuse on Apache Karaf, make sure your system meets the minimum requirements.

SUPPORTED PLATFORMS

Red Hat tests and supports Fuse products in the configurations listed at Red Hat JBoss Fuse Supported Configurations.

JAVA RUNTIME

For details of the Java runtimes supported by Fuse on Apache Karaf, see Red Hat JBoss Fuse Supported Configurations.

WARNING

Do **not** install the Java runtime under a directory path that includes whitespace. For example, `C:\Program Files\Java\jdk8` is not an acceptable install path and will lead to unpredictable errors in Fuse on Apache Karaf at run time.

SUPPORTED STANDARDS

Fuse on Apache Karaf supports the standards and protocols listed at Red Hat JBoss Fuse Supported Standards.

HARDWARE REQUIREMENTS

The minimum hardware requirements for installing a full installation of Fuse on Apache Karaf are:

- 250 MB of free disk space
- 2 GB of RAM

In addition to the disk space required for the base installation, a running system will require space for caching, persistent message stores, and other functions.

RECOMMENDED SOFTWARE

It is recommended that you use Maven with Fuse on Apache Karaf projects. For information about preparing to use Maven, see Appendix B, *Preparing to use Maven*.

CHAPTER 2. INSTALLATION TYPES

The standard install package for Fuse 7.1 on Karaf is available for download from the Red Hat Customer Portal. It installs the standard assembly of the Apache Karaf container, including the full Fuse technology stack.

It is possible to create your own **custom assembly** of Fuse 7.1, containing a customized subset of the Fuse features and bundles.

CHAPTER 3. INSTALLING ON APACHE KARAF

Red Hat Fuse is installed by unpacking an archive file on your file system.

GETTING THE ARCHIVE

You can download the Fuse on Karaf archive from the Red Hat Customer Portal→Downloads→Red Hat JBoss Middleware→Downloads page, after you register and log in to your customer account.

Once logged in:

1. Select **Fuse**, listed under **Integration Platforms**, in the sidebar menu.
2. Select **7.1.0** from the **Version** drop-down list on the **Software Downloads** page.
3. Click the **Download** button next to the *Red Hat Fuse 7.1.0 on Karaf Installer*file.

UNPACKING THE ARCHIVE

Fuse on Karaf is packaged as a **.zip** file. Using a suitable archive tool, such as **Zip**, unpack Fuse on Karaf into a directory to which you have full access.

WARNING

Do not unpack the archive file into a folder that has spaces in its path name. For example, do not unpack into **C:\Documents and Settings\Greco Roman\Desktop\fuse**.

WARNING

Do not unpack the archive file into a folder that has any of the following special characters in its path name: **#**, **%**, **^**, **"**.

USING THE IBM JDK

If you are using the IBM JDK, remove the **saaj-api** jar from the ***installDir*/lib/endorsed** library using the following command:

```
rm lib/endorsed/org.apache.servicemix.specs.saaj-api-1.3-2.7.0.jar
```

Before invoking the **./bin/fuse** script:, set the **JAVA_OPTS** environment variable as follows:

```
JAVA_OPTS=-Xshareclasses:none
```

CHAPTER 4. ADDING A REMOTE CONSOLE USER

Fuse on Karaf is not installed with a default user for the remote console. You must add a user before you can connect to the server's remote console. To add a user, edit **`InstallDir/etc/users.properties`**.

IMPORTANT

The information in this file is unencrypted so it is not suitable for environments that require strict security.

To add a user:

1. Open **`InstallDir/etc/users.properties`** in your favorite text editor.
2. Locate the following lines:

```
#admin = admin,_g_:admingroup
#_g_\:admingroup = group,admin,manager,viewer,systembundles,ssh
```

   Note that the first line has the syntax **`USER=PASSWORD,_g_:GROUP,ROLE1,ROLE2,...`**. In this example, the first line specifies a user, **`admin`**, with the password, **`admin`**, and the role group, **`admingroup`**.

3. Uncomment both lines by removing the leading # character.
4. Change the first **`admin`** to the desired user name.
5. Change the second **`admin`** to the desired password.
6. Save the changes.

CHAPTER 5. OFFLINE MODE

You can run the Apache Karaf container in offline mode (that is, without an Internet connection). But if you are planning to deploy custom applications to the container, it might be necessary to download additional dependencies to a local Maven repository before you can run the container in offline mode with these applications.

To run the Apache Karaf container in offline mode, it is necessary to distinguish between the following kinds of dependency:

- **Runtime dependencies** — the dependencies required to run the Apache Karaf container, in its default configuration.
- **Build-time dependencies** — the dependencies required to build a custom application (which might include third-party libraries).

Here is a summary of what can be done in offline mode and what needs to be done in online mode (with an Internet connection):

- **Running the Apache Karaf container in its default configuration** — is supported in offline mode. The default configuration of the Apache Karaf container is specified by the **`featuresBoot`** property in the **`etc/org.apache.karaf.features.cfg`** file. The requisite dependencies are all provided in the **`system/`** sub-directory of the installation.
- **Installing additional features** — is, in general, **not** supported in offline mode. In principle, you can use the **`features:install`** command to install any of the features from the standard feature repositories (as specified by the **`featuresRepositories`** property in the **`etc/org.apache.karaf.features.cfg`** file), but the majority of these features must be downloaded from the Internet and are thus not supported in offline mode.
- **Deploying custom applications** — is, in general, **not** supported in offline mode. There may be some cases where an application with a minimal set of build-time dependencies is deployable offline, but in general, custom applications would have third-party dependencies that require an Internet connection (so that JAR files can be downloaded by Apache Maven).

If you do need to deploy an application with dependencies that are not available offline, you can use the Maven dependency plug-in to download the application's dependencies into a Maven offline repository. This customized Maven offline repository can then be distributed internally to any machines that do not have an Internet connection.

5.1. DOWNLOAD REPOSITORY FOR MAVEN PROJECT

From the project directory that contains the **`pom.xml`** file, run the following Maven command:

```
mvn org.apache.maven.plugins:maven-dependency-plugin:3.1.0:go-offline -
Dmaven.repo.local=/tmp/foo
```

All the Maven dependencies and plug-ins required to build the project will be downloaded to the **`/tmp/foo`** directory.

5.2. ENABLE OFFLINE MODE

To enable offline mode you must edit **`etc/org.ops4j.pax.url.mvn.cfg`**. Find the setting **`org.ops4j.pax.url.mvn.offline`** and replace the default **`false`** with **`true`**.

```
##
# If set to true, no remote repository will be accessed when resolving
artifacts
#
org.ops4j.pax.url.mvn.offline = true
```

CHAPTER 6. APPLYING PATCHES

Abstract

Red Hat Fuse supports incremental patching. FuseSource will supply you with easy to install patches that only make targeted changes to a deployed container.

6.1. PATCHING OVERVIEW

Patching enables you apply fixes to a Karaf container without needing to reinstall an updated version of Fuse on Karaf. It also allows you to back out the patch, if it causes problems with your deployed applications.

Patches are ZIP files that contain the artifacts needed to update a targeted set of bundles in a container. The artifacts are typically one or more bundles. They can, however, include configuration files and feature descriptors.

You get a patch file in one of the following ways:

- Customer Support sends you a patch.
- Customer Support sends you a link to download a patch.
- Download a patch directly from the Red Hat customer portal.

The process of applying a patch to a container depends on how the container is deployed:

- **Standalone (standard process)** — using commands from the Karaf console's **`patch`** shell. This approach is non-destructive and reversible.

6.2. FINDING THE RIGHT PATCHES TO APPLY

Abstract

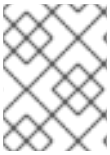
This section explains how to find the patches for a specific version of Fuse on the Red Hat Customer Portal and how to figure out which patches to apply, and in what order.

Locate the patches on the customer portal

If you have a subscription for Fuse, you can download the latest patches directly from the Red Hat Customer Portal. Locate the patches as follows:

1. Login to the Red Hat Customer Portal using your customer account. This account **must** be associated with an appropriate Red Hat software subscription, otherwise you will not be able to see the patch downloads for Fuse.
2. Navigate to the customer portal Software Downloads page.
3. In the **`Product`** dropdown menu, select Red Hat Fuse and then select the version, 7.1, from the **`Version`** dropdown menu. A table of downloads now appears, which has three tabs: **`Releases`**, **`Patches`**, and **`Security Advisories`**.
4. Click the **`Releases`** tab to view the GA product releases.

5. Click the **Patches** tab to view the rollup patches and the regular incremental patches (with no security-related fixes).
6. Click the **Security Advisories** tab to view the incremental patches with security-related fixes.



NOTE

To see the **complete** set of patches, you must look under the **Releases** tab, the **Patches** tab **and** the **Security Advisories** tab.

Types of patch

The following types of patch can be made available for download:

- Rollup patches
- Incremental patches

Rollup patches

A rollup patch is a cumulative patch that incorporates **all** of the fixes from the preceding patches. Moreover, each rollup patch is regression tested and establishes a new baseline for the application of future patches.

In Fuse, a rollup patch file is dual-purpose, as follows:

- Each rollup patch file is a complete new build of the official target distribution. This means you can unzip the rollup patch file to obtain a completely new installation of Fuse, just as if it was a fresh download of the product (which, in fact, it is). See Section 6.3, “Installing a Rollup Patch as a New Installation”.
- You can also treat the rollup patch as a regular patch, using it to upgrade an existing installation. That is, you can provide the rollup patch file as an argument to the standalone patch console commands (for example, **patch:add** and **patch:install**) or the Fabric patch console command, **patch:fabric-install**.

Incremental patches

Incremental patches are patches released either directly after GA or after a rollup patch, and they are intended to be applied on top of the corresponding build of Fuse. The main purpose of an incremental patch is to update some of the bundles in an existing distribution.

Which patches are needed to update the GA product to the latest patch level?

To figure out which patches are needed to update the GA product to the latest patch level, you need to pay attention to the type of patches that have been released so far:

1. If the only patches released so far are patches with GA baseline (Patch 1, Patch 2, and so on), apply the **latest** of these patches directly to the GA product.
2. If a rollup patch has been released and no patches have been released after the latest rollup patch, simply apply the latest rollup patch to the GA product.

3. If the latest patch is a patch with a rollup baseline, you must apply two patches to the GA product, as follows:

   a. Apply the latest rollup patch, and then

   b. Apply the latest patch with a rollup baseline.

Which patches to apply, if you only want to install regression-tested patches?

If you prefer to install only patches that have been regression tested, install the latest rollup patch.

6.3. INSTALLING A ROLLUP PATCH AS A NEW INSTALLATION

A rollup patch is a new build

In Fuse, a rollup patch file is a complete new build of the official target distribution. In other words, it is just like the original GA distribution, except that it includes later build artifacts.

Installing the new build

To install a new build, corresponding to a rollup patch level, perform the following steps:

1. Identify which rollup patch you need to install and download it from the Customer Portal. For more details, see Section 6.2, “Finding the Right Patches to Apply”.

2. Unzip the rollup patch file to a convenient location, just as you would with a regular GA distribution. This is your new installation of Fuse.

Comparison with patch process

Compared with the conventional patch process, installing a new build has the following advantages and limitations:

- This approach is only for creating a completely new installation of Fuse. If your existing installation already has a lot of custom configuration, this might not be the most convenient approach to use.

- The new build includes only the artifacts and configuration for the new patch level.

6.4. PATCHING A STANDALONE CONTAINER

Abstract

You apply patches to a standalone container using the command console’s **`patch`** shell. You can apply and roll back patches as needed.

Overview

When patching a standalone container, you can apply either an incremental patch or a rollup patch. There are very significant differences between the two kinds of patch and the way they are applied. Although the same commands are used in both cases, the internal processes are different (the patch commands auto-detect the patch type).

Incremental patch

An incremental patch is used mainly to update the **bundle JARs** in the container. This type of patch is suitable for delivering hot fixes to the Fuse installation, but it has its limitations. An incremental patch:

- Updates bundle JARs.
- Patches only the current container instance (under the **`data/`** directory). Hence, patches are **not** preserved after deleting a container instance.
- Updates any feature dependencies installed in the current container instance, but does not update the feature files themselves.
- Might update some configuration files, but is **not** suitable for updating most configuration files.
- Supports patch rollback.

After applying an incremental patch to a standalone container, meta-data about the patch is written to the **`etc/startup.properties`** and **`etc/overrides.properties`** files. With these files, the Karaf installation is able to persist the patch even after deleting the root container instance (that is, after removing the root container's **`data/`** directory).

NOTE

Removing the **`data/cache`** directory uninstalls any bundles, features, or feature repositories that were installed into the container using Karaf console commands. You can remove the **`data/cache`** directory only after you stop the container. However, any patches that have been applied will remain installed, as long as the **`etc/startup.properties`** and **`etc/overrides.properties`** files are preserved.

Rollup patch

A rollup patch can make updates to **any** installation files including **bundle JARs** and **static files** (including, for example, configuration files under the **`etc/`** directory). A rollup patch:

- Updates any files, including bundle JARs, configuration files, and any static files.
- Patches both the current container instance (and its runtime storage under the **`data/`** directory) and the underlying installation. Hence, patches are preserved after deleting a container instance.
- Updates all of the files related to Karaf features, including the features repository files and the features themselves. Hence, any features installed after the rollup patch will reference the correct patched dependencies.
- If necessary, updates configuration files (for example, files under **`etc/`**), automatically merging any configuration changes you have made with the configuration changes made by the patch. If merge conflicts occur, see the patch log for details of how they are handled.
- Most of the merge conflicts are resolved automatically. For example, the patch mechanism detects conflicts at property level for the property files. It detects whether it was a user or patch that changed any property. The change is preserved, if only one side changed the property.
- Tracks **all** of the changes made to the installation (including to static files), so that it is possible to roll back the patch.

NOTE

The rollup patching mechanism uses an internal git repository (located under **`patches/.management/history`**) to track the changes made.

Patching the patch mechanism

(Recommended, if applicable) If there is no patch management package corresponding to the rollup patch you are about to install, then you can skip this procedure and install the rollup patch directly.

From time to time, important changes and improvements are made to the patch mechanism. In order to pick up these improvements, we recommend that you patch the patch mechanism to a higher level **before** upgrading Fuse with a rollup patch. If you were to upgrade straight to the latest rollup patch version of Fuse, the improved patch mechanism would become available **after** you completed the upgrade. But at that stage, it would be too late to benefit from the improvements in the patch mechanism.

To circumvent this bootstrap problem, the improved patch mechanism is made available as a separate download, so that you can patch the patch mechanism itself, **before** you upgrade to the new patch level.

NOTE

The **`fuse-patch-management-<version>.zip`** archive is used to quickly patch the patching mechanism.

To patch the patch mechanism, proceed as follows:

1. Download the appropriate patch management package. From the download page, select a package named **`Red Hat Fuse 7.x.x Rollup N on Karaf Update Installer`**, where ***`N`*** is the number of the particular rollup patch you are about to install.

2. Install the patch management package on top of your existing installation. Use an archive utility to extract the contents on top of the existing Karaf container installation (installing files under the **`system/`** subdirectory).

NOTE

It does not matter whether the container is running or not when you extract these files.

3. Start the container, if it is not already running.

4. Run the **`patch:update`** command. There is no need to restart the container.

```
karaf@root()> la -l|grep patch

233 | Active   |  80 | 7.0.0.fuse-000191-redhat-1    |
mvn:org.jboss.fuse.modules.patch/patch-commands/7.0.0.fuse-000191-redhat-1
234 | Active   |  35 | 7.0.0.fuse-000191-redhat-1    |
mvn:org.jboss.fuse.modules.patch/patch-core/7.0.0.fuse-000191-redhat-1
235 | Active   |  35 | 7.0.0.fuse-000191-redhat-1    |
mvn:org.jboss.fuse.modules.patch/patch-core-api/7.0.0.fuse-000191-redhat-1
236 | Active   |   2 | 7.0.0.fuse-000191-redhat-1    |
mvn:org.jboss.fuse.modules.patch/patch-management/7.0.0.fuse-000191-
redhat-1
```

```
karaf@root()> patch:update --simulation

Current patch mechanism version: 7.0.0.fuse-000191-redhat-1
New patch mechanism version detected: 7.0.1.fuse-000011-redhat-3

karaf@root()> patch:update

Current patch mechanism version: 7.0.0.fuse-000191-redhat-1
New patch mechanism version detected: 7.0.1.fuse-000011-redhat-3
Uninstalling patch features in version 7.0.0.fuse-000191-redhat-1
Installing patch features in version 7.0.1.fuse-000011-redhat-3

karaf@root()> la -l|grep patch

237 | Active   |  80 | 7.0.1.fuse-000011-redhat-3    |
mvn:org.jboss.fuse.modules.patch/patch-commands/7.0.1.fuse-000011-redhat-3
238 | Active   |  35 | 7.0.1.fuse-000011-redhat-3    |
mvn:org.jboss.fuse.modules.patch/patch-core/7.0.1.fuse-000011-redhat-3
239 | Active   |  35 | 7.0.1.fuse-000011-redhat-3    |
mvn:org.jboss.fuse.modules.patch/patch-core-api/7.0.1.fuse-000011-redhat-3
240 | Active   |   2 | 7.0.1.fuse-000011-redhat-3    |
mvn:org.jboss.fuse.modules.patch/patch-management/7.0.1.fuse-000011-
redhat-3

karaf@root()> patch:update

Current patch mechanism version: 7.0.1.fuse-000011-redhat-3
No newer version of patch bundles detected
```

Applying a patch

To apply a patch to a standalone container:

1. Make a full backup of your Fuse installation before attempting to apply the patch.
2. **(Rollup patch only)** Before applying the rollup patch to your container, you **must** patch the patch mechanism, as described in the section called “Patching the patch mechanism”.
3. **(Incremental patch only)** Before you proceed to install the patch, make sure to read the text of the **`README`** file that comes with the patch, as there might be additional **manual steps** required to install a particular patch.
4. Start the container, if it is not already running. If the container is running in the background (or remotely), connect to the container using the SSH console client, **`bin/client`**.
5. Add the patch to the container's environment by invoking the **`patch:add`** command. For example, to add the ***`patch`*`.zip`** patch file:

   ```
   patch:add file://patch.zip
   ```

6. Simulate installing the patch by invoking the **`patch:simulate`** command.
   This generates a log of the changes that will be made to the container when the patch is installed, but will not make any actual changes to the container. Review the simulation log to understand the changes that will be made to the container.

7. Invoke the **patch:list** command to display a list of added patches. In this list, the entries under the **[name]** heading are patch IDs. For example:

```
karaf@root()> patch:list

[name]  [installed] [rollup] [description]

fuse-karaf-7.1.0.fuse-710018-redhat-00001 false  true  fuse-karaf-
7.1.0.fuse-710018-redhat-00001
```

   Ensure that the container has fully started before you try to perform the next step. In some cases, the container must restart before you can apply a patch, for example, if static files are patched. In these cases, the container restarts automatically.

8. Apply a patch to the container by invoking the **patch:install** command and specifying the patch ID for the patch that you want to apply. For example:

```
karaf@root()> patch:install fuse-karaf-7.1.0.fuse-710018-redhat-
00001
```

9. Validate the patch, by searching for one of the patched artifacts. For example, if you had just upgraded Fuse 7.1.0 to the patch with build number **N**, you could search for bundles with this build number, as follows:

```
karaf@root()> la -l|grep 710018

  2 | Active   |   2 | 7.1.0.fuse-710018        |
mvn:org.jboss.fuse.modules.patch/patch-management/7.1.0.fuse-710018

 22 | Active   |  80 | 7.1.0.fuse-710018        |
mvn:org.jboss.fuse.modules/fuse-pax-transx-tm-narayana/7.1.0.fuse-
710018

 24 | Active   |  80 | 2.0.0.fuse-710018        |
mvn:io.hawt/hawtio-log/2.0.0.fuse-710018

 25 | Active   |  80 | 2.0.0.fuse-710018        |
mvn:io.hawt/hawtio-log-osgi/2.0.0.fuse-710018

 26 | Active   |  80 | 2.0.0.fuse-710018        |
mvn:io.hawt/hawtio-osgi-jmx/2.0.0.fuse-710018
```

After applying a rollup patch, you also see the new version and build number in the Welcome banner when you restart the container.

Rolling back a patch

Occasionally a patch will not work or will introduce new issues to a container. In these cases, you can easily back the patch out of the system and restore it to pre-patch behaviour using the **patch:rollback** command, as follows:

1. Invoke the **patch:list** command to obtain the patch ID, ***PatchID***, of the most recently installed patch.

2. Invoke the **patch:rollback** command, as follows:

```
patch:rollback PatchID
```

In some cases the container will need to restart to roll back the patch. In these cases, the container restarts automatically. Due to the highly dynamic nature of the OSGi runtime, during the restart you might see some occasional errors related to incompatible classes. These are related to OSGi services that have just started or stopped. These errors can be safely ignored.

APPENDIX A. INSTALLING THE APACHE COMPONENTS

Red Hat Fuse provides an additional package to download, which contains the standard distributions of Apache Camel and Apache CXF. If you want to use a standard distribution of Apache Camel or Apache CXF (without the OSGi container) use the archived versions in the downloaded extras package.

A.1. GETTING THE EXTRAS ARCHIVE

You can download the extras archive from the Red Hat Customer Portal→Downloads→Red Hat JBoss Middleware→Downloads page, after you register and log in to your customer account.

Once logged in:

1. Select **Fuse**, listed under **Integration Platforms**, in the sidebar menu.
2. Select **7.1.0** from the **Version** drop-down list on the **Software Downloads** page.
3. Download **fuse-extras-7.1.0.fuse-710018-redhat-00001.zip** archive.

A.2. CONTENTS OF THE EXTRAS ARCHIVE

The extras archive file contains the following archive files nested inside it:

- **apache-camel-2.21.0.fuse-710018-redhat-00001.zip**
- **apache-cxf-3.1.11.fuse-710022-redhat-00001.zip**

You can copy these files to the desired location and decompress them using the appropriate utility for your platform.

WARNING

Do not unpack an archive file into a folder that has spaces in its path name. For example, do not unpack into **C:\Documents and Settings\Greco Roman\Desktop\fuse**.

APPENDIX B. PREPARING TO USE MAVEN

B.1. OVERVIEW

This section gives a brief overview of how to prepare Maven for building Red Hat Fuse projects and introduces the concept of Maven coordinates, which are used to locate Maven artifacts.

B.2. PREREQUISITES

In order to build a project using Maven, you must have the following prerequisites:

- **Maven installation** — Maven is a free, open source build tool from Apache. You can download the latest version from the Maven download page.
- **Network connection** — whilst performing a build, Maven dynamically searches external repositories and downloads the required artifacts on the fly. By default, Maven looks for repositories that are accessed over the Internet. You can change this behavior so that Maven will prefer searching repositories that are on a local network.

NOTE

Maven can run in an offline mode. In offline mode Maven only looks for artifacts in its local repository.

B.3. ADDING THE RED HAT MAVEN REPOSITORIES

In order to access artifacts from the Red Hat Maven repositories, you need to add them to Maven's **`settings.xml`** file. Maven looks for your **`settings.xml`** file in the **`.m2`** directory of the user's home directory. If there is not a user specified **`settings.xml`** file, Maven will use the system-level **`settings.xml`** file at **`M2_HOME/conf/settings.xml`**.

To add the Red Hat repositories to Maven's list of repositories, you can either create a new **`.m2/settings.xml`** file or modify the system-level settings. In the **`settings.xml`** file, add **`repository`** elements for the Red Hat repositories as shown in Adding the Red Hat Fuse Repositories to Maven.

Adding the Red Hat Fuse Repositories to Maven

```
<?xml version="1.0"?>
<settings>

  <profiles>
    <profile>
      <id>extra-repos</id>
      <activation>
        <activeByDefault>true</activeByDefault>
      </activation>
      <repositories>
       <repository>
            <id>redhat-ga-repository</id>
            <url>https://maven.repository.redhat.com/ga</url>
            <releases>
                <enabled>true</enabled>
```

```
            </releases>
            <snapshots>
                <enabled>false</enabled>
            </snapshots>
        </repository>
        <repository>
            <id>redhat-ea-repository</id>

<url>https://maven.repository.redhat.com/earlyaccess/all</url>
            <releases>
                <enabled>true</enabled>
            </releases>
            <snapshots>
                <enabled>false</enabled>
            </snapshots>
        </repository>
        <repository>
          <id>jboss-public</id>
          <name>JBoss Public Repository Group</name>

<url>https://repository.jboss.org/nexus/content/groups/public/</url>
        </repository>
      </repositories>
      <pluginRepositories>
        <pluginRepository>
            <id>redhat-ga-repository</id>
            <url>https://maven.repository.redhat.com/ga</url>
            <releases>
                <enabled>true</enabled>
            </releases>
            <snapshots>
                <enabled>false</enabled>
            </snapshots>
        </pluginRepository>
        <pluginRepository>
            <id>redhat-ea-repository</id>

<url>https://maven.repository.redhat.com/earlyaccess/all</url>
            <releases>
                <enabled>true</enabled>
            </releases>
            <snapshots>
                <enabled>false</enabled>
            </snapshots>
        </pluginRepository>
        <pluginRepository>
          <id>jboss-public</id>
          <name>JBoss Public Repository Group</name>

<url>https://repository.jboss.org/nexus/content/groups/public</url>
        </pluginRepository>
      </pluginRepositories>
    </profile>
  </profiles>

  <activeProfiles>
```

```
    <activeProfile>extra-repos</activeProfile>
  </activeProfiles>

</settings>
```

B.4. ARTIFACTS

The basic building block in the Maven build system is an *artifact*. The output of an artifact, after performing a Maven build, is typically an archive, such as a JAR or a WAR.

B.5. MAVEN COORDINATES

A key aspect of Maven functionality is the ability to locate artifacts and manage the dependencies between them. Maven defines the location of an artifact using the system of *Maven coordinates*, which uniquely define the location of a particular artifact. A basic coordinate tuple has the form, **{*groupId*, *artifactId*, *version*}**. Sometimes Maven augments the basic set of coordinates with the additional coordinates, *packaging* and *classifier*. A tuple can be written with the basic coordinates, or with the additional *packaging* coordinate, or with the addition of both the *packaging* and *classifier* coordinates, as follows:

```
groupId:artifactId:version
groupId:artifactId:packaging:version
groupId:artifactId:packaging:classifier:version
```

Each coordinate can be explained as follows:

groupdId

Defines a scope for the name of the artifact. You would typically use all or part of a package name as a group ID — for example, **`org.fusesource.example`**.

artifactId

Defines the artifact name (relative to the group ID).

version

Specifies the artifact's version. A version number can have up to four parts: **`n.n.n.n`**, where the last part of the version number can contain non-numeric characters (for example, the last part of **`1.0-SNAPSHOT`** is the alphanumeric substring, **`0-SNAPSHOT`**).

packaging

Defines the packaged entity that is produced when you build the project. For OSGi projects, the packaging is **`bundle`**. The default value is **`jar`**.

classifier

Enables you to distinguish between artifacts that were built from the same POM, but have different content.

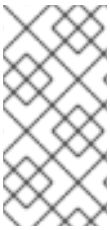
The group ID, artifact ID, packaging, and version are defined by the corresponding elements in an artifact's POM file. For example:

```
<project ... >
  ...
  <groupId>org.fusesource.example</groupId>
  <artifactId>bundle-demo</artifactId>
  <packaging>bundle</packaging>
  <version>1.0-SNAPSHOT</version>
```

```
  ...
</project>
```

For example, to define a dependency on the preceding artifact, you could add the following **dependency** element to a POM:

```
<project ... >
  ...
  <dependencies>
    <dependency>
      <groupId>org.fusesource.example</groupId>
      <artifactId>bundle-demo</artifactId>
      <version>1.0-SNAPSHOT</version>
    </dependency>
  </dependencies>
  ...
</project>
```



NOTE

It is **not** necessary to specify the **`bundle`** package type in the preceding dependency, because a bundle is just a particular kind of JAR file and **`jar`** is the default Maven package type. If you do need to specify the packaging type explicitly in a dependency, however, you can use the **`type`** element.