OpenEngSB Maven Plugin Manual

Version 2.3.1

Table of Contents

Part I. User Manual

The user manual explains how the OpenEngSB Maven Plugin is to be used.

Chapter 1. Using the OpenEngSB Maven Plugin

The openengsb-maven-plugin is a plugin for Apache Maven, intended to simplify various activities (creating domains or connectors, building a release artifact of the whole project etc.) when developing based on the OpenEngSB.

1.1. Purpose of the openengsb-maven-plugin

The purpose of the OpenEngSB Maven Plugin is to provide additional useful goals for the development of the OpenEngSB itself and all projects which base on the OpenEngSB. It helps in various goals starting in assembling, checkstyle, license checking and many other various goals which would otherwise require to duplicate tons of version (and manage it in addition) between the OpenEngSB and projects which are based on the OpenEngSB.

1.2. Configuring the openengsb-maven-plugin for your project

To use the openengsb-maven-plugin in your project add the following configuration to your project's pom.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instanc
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
  ...
  <build>
    <plugins>
      <plugin>
        <groupId>org.openengsb</groupId>
        <artifactId>openengsb-maven-plugin</artifactId>
        <version>${openengsb.maven.plugin.version}</version>
      </plugin>
    </plugins>
  </build>
  ...
</project>
```

The plugin can now be invoked via **mvn openengsb:<goal>**

1.3. Available Goals

assemble or `etc/scripts/assemble.sh`

Installs the OpenEngSB and skips tests. Furthermore a *nightly* profile is activated if available in your poms.

eclipse or `etc/scripts/eclipse.sh`

Generates eclipse configuration file for the module where it is invoked from and all submodules. The generated eclipse projects are also configured to use the checkstyle rules shipped with the plugin (see checkstyle mojo).

checkstyle

Performs a Checkstyle check of the project. The checkstyle configuration file which is used for the check can be found here. We ship this configuration file with the plugin (and changes need to be done there) because we think it may be useful for other OpenEngSB related projects. Setting up eclipse projects with configured checkstyle becomes very easy that way (simply mvn openengsb:eclipse).

genConnector or `etc/scripts/gen-connector.sh` (For additional info how to create a connector see ???)

Guides interactively through the creation of a connector and generates the artifact via the connector archetype.

genDomain or `etc/scripts/gen-domain.sh` (For additional info how to create a domain see ???)

Guides interactively through the creation of a domain and generates the artifact via the domain archetype.

licenseCheck or `etc/scripts/license-check.sh`

Performs a check if appropriate license headers are available in every source file. The licenseCheck mojo wraps the com.mycila.maven-license-plugin. A large part of the default behavior of this mojo can be changed in `src/main/resources/license/licenseConfig.xml`. See this site for available configuration options. The openengsb-maven-plugin needs to be reinstalled after changing its default behavior.

NOTE: pom.xml files are excluded from license check

Parameters:

- *additionalExcludes*

  Defines path to a file where each line represents a pattern which files to exclude from license check or license format (additionally to the default excludes).

licenseFormat or `etc/scripts/license-format.sh`

Adds a license header to files where the license header is missing. Regarding the configuration for this mojo the same applies as for licenseCheck.

NOTE: pom.xml files are excluded from license format

Parameters:

- *additionalExcludes*

  see description of licenseCheck

prePush or `etc/scripts/pre-push.sh`

Builds and installs the openengsb, checks for license headers, performs a Checkstyle check and runs the integration tests.

Parameters:

- *additionalExcludes*

  see description of licenseCheck

provision or `etc/scripts/run.sh` / `etc/scripts/quickrun.sh`

Equivalent to execute karaf or karaf.bat per hand after build by mvn clean install in a (typically) assembly directory.

Parameters:

- *provisionPathUnix*

  This setting should be done in the one of the assembly folders and have to point to the final directory where the karaf system, etc configs and so on consist.

- *provisionExecutionPathUnix*

  The path to the executable in the unix archive file

- *additionalRequiredExecutionPathUnix*

  Sometimes it's required that some executable files, provided in *provisionExecutionPathUnix* execute other files which have to made executable to work correctly on themselves. Those files should be specified here.

- *provisionPathWindows*

  This setting should be done in the one of the assembly folders and have to point to the final directory where the karaf system, etc configs and so on consist.

- *provisionExecutionPathWindows*

  The path to the executable in the windows archive file

- *additionalRequiredExecutionPathWindows*

  Sometimes it's required that some executable files, provided in *provisionExecutionPathWindows* execute other files which have to made executable to work correctly on themselves. Those files should be specified here.

These parameters are typically configured in the pom of your assembly project (`/assembly/pom.xml` for the OpenEngSB)).

pushVersion or `etc/scripts/push-version.sh`

Updates the development version.

Parameters:

- *developmentVersion*

  The new SNAPSHOT version.

extractSource

The goal is a really powerful for including source code into the documentation. It recursively scans all files on a defined path for specific comments in the code and extracts the source in between into a soruce listing which could be included afterwards easily. Currently the plugin scans the following files: .java, .xml, .properties and .cfg. To start an exclude in java your code needs to look like...

```
    // @extract-start javaout
    private App() {
    }
    // @extract-end
```

... The format of the comments have to be eactly of the format as shown in the sample. "javaout" could be replaced on the other hand with whatever you like. A file will be created in the target folder with the name of your choice ("javaout" in this case) containing the content between the comments. In addition the "programlisting" and the correct language tag is attached. This allows to directly include the code which is also compiled for your project, to be included into the documentation.

For xml a simple example would look like:

```
<blueprint>

  <!-- @extract-start xmlout -->
  <service id="abc" interface="a.b.c">
    <bean class="a.b.d" />
  </service>
  <!-- @extract-end -->

</blueprint>
```

Property files and cfg files are of exactly the same format and would need to look like:

```
# @extract-start propout
timetablePageName=Timetable
# @extract-end
```

Parameters:

- *sourcePath*

  The path which should be scanned for sources.

- *targetPath*

  The path where the generated files should be pushed to.

releaseNightly or `etc/scripts/release-nightly.sh`

Mojo to perform nightly releases. This mojo activates the *nightly* profile in the project, where you can put your additional configuration for nightly releases (To see what these profiles can be necessary for please read the descript of the other release mojos).

release<XXX> (You can find a detailed description of the OpenEngSB release process in ???)

The release<XXX> mojos (except Nightly) wrap the maven-license-plugin, basically just invoking `mvn release:prepare` and then `mvn release:perforn` with some useful default configuration which can be reused for other projects related to the openengsb. These mojos perform a release and activate the <*XXX*> profile. These release profiles are important and are required to ..

- .. select the appropriate passphrase for the maven release repository from your `settings.xml`. For additional information on this topic see ???.
- .. set links depending on the release type. For examples please see the profiles in the pom
- .. configure distribution management of the project site, depending on the release type. For examples see profiles in docs/homepage/pom

Parameters:

- *connectionUrl*

  URL to your SCM repository (e.g. scm:git:file://~/openengsb). During the release process changes (version updates, etc) are commited into your SCM.

Goals:

- **releaseFinal** or `etc/scripts/release-final.sh`

  profile = *final*
- **releaseMilestone** or `etc/scripts/release-milestone.sh`

  profile = *milestone*
- **releaseRC** or `etc/scripts/release-rc.sh`

  profile = *rc*
- **releaseSupport** or `etc/scripts/release-support.sh`

  profile = *support*

Part II. Contributor Manual

The contributor manual explains the internal parts of the OpenEngSB Maven Plugin to show where and how it could be best extended.

Chapter 2. OpenEngSB Maven Plugin for Contributor

While the previous chapter gave a detailed description of the maven plugin this one focues on the internals of the plugin and tries to explain the internal structure, helping ppl who want to understand the internals and extend it.

2.1. Exract Sources for Documentation

Sources for documentation are extracted using the org.openengsb.openengsbplugin.ExtractDocSourceMojo. The mojo itself includes a list of implementations of the org.openengsb.openengsbplugin.extract.AnnotatedSourceExtractor interface.

```
/**
 * Base interface which needs to be implemented to add an additional source type to analyse.
 */
public interface AnnotatedSourceExtractor {

    /**
     * Checks if the extractor can handle a specific file type.
     */
    boolean canExtractFile(File file);

    /**
     * Checks if the line is a start line for the extractor.
     */
    boolean isStartLine(String line);

    /**
     * Checks if the specific line is a stop line for the extractor.
     */
    boolean isStopLine(String line);

    /**
     * Since the lines could be structured quite specific the extrator need to extract the target file
     * name.
     */
    public String extractTargetFilenameFromLine(String line);

    /**
     * This one is responsible for code highlighting. A java extractor might return java here while a
     * return xml here.
     */
    String getLanguage();

}
```