

HP Operations Orchestration

For Windows and Linux

Software Version: 10.00

Extension Developers Guide

Document Release Date: June 2013

Software Release Date: June 2013



Legal Notices

Warranty

The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be liable for technical or editorial errors or omissions contained herein.

The information contained herein is subject to change without notice.

Restricted Rights Legend

Confidential computer software. Valid license from HP required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Copyright Notice

© Copyright 2013 Hewlett-Packard Development Company, L.P.

Trademark Notices

Adobe™ is a trademark of Adobe Systems Incorporated.

This product includes an interface of the 'zlib' general purpose compression library, which is Copyright © 1995-2002 Jean-loup Gailly and Mark Adler.

AMD and the AMD Arrow symbol are trademarks of Advanced Micro Devices, Inc.

Google™ and Google Maps™ are trademarks of Google Inc.

Intel®, Itanium®, Pentium®, and Intel® Xeon® are trademarks of Intel Corporation in the U.S. and other countries.

Java is a registered trademark of Oracle and/or its affiliates.

Microsoft®, Windows®, Windows NT®, Windows® XP, and Windows Vista® are U.S. registered trademarks of Microsoft Corporation.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates.

UNIX® is a registered trademark of The Open Group.

Documentation Updates

The title page of this document contains the following identifying information:

- Software Version number, which indicates the software version.
- Document Release Date, which changes each time the document is updated.
- Software Release Date, which indicates the release date of this version of the software.

To check for recent updates or to verify that you are using the most recent edition of a document, go to:

<http://h20230.www2.hp.com/selfsolve/manuals>

This site requires that you register for an HP Passport and sign in. To register for an HP Passport ID, go to:

<http://h20229.www2.hp.com/passport-registration.html>

Or click the **New users - please register** link on the HP Passport login page.

You will also receive updated or new editions if you subscribe to the appropriate product support service. Contact your HP sales representative for details.

Support

Visit the HP Software Support Online web site at:

<http://www.hp.com/go/hpsoftwaresupport>

This web site provides contact information and details about the products, services, and support that HP Software offers.

HP Software online support provides customer self-solve capabilities. It provides a fast and efficient way to access interactive technical support tools needed to manage your business. As a valued support customer, you can benefit by using the support web site to:

- Search for knowledge documents of interest
- Submit and track support cases and enhancement requests
- Download software patches
- Manage support contracts
- Look up HP support contacts
- Review information about available services
- Enter into discussions with other software customers
- Research and register for software training

Most of the support areas require that you register as an HP Passport user and sign in. Many also require a support contract. To register for an HP Passport ID, go to:

<http://h20229.www2.hp.com/passport-registration.html>

To find more information about access levels, go to:

http://h20230.www2.hp.com/new_access_levels.jsp

Contents

Extension Developers Guide	1
Contents	5
Developing Extensions for HP OO	7
Creating an @Action	8
Developing Plugins	8
Developing Extensions	11
"Hello World!" example	11
Passing Arguments to @Actions	11
Return Values	12
Adding @Action Annotations	12
Annotations	12
@Action Data Definition Example	15
Testing Extensions	15
Testing Extensions as Part of the Project Build	15
Testing Extensions Independently from the Command Line	15
.NET Extensions	17
Legacy Actions	21

Disclaimer for PDF Version of Online Help

This document is a PDF version of the online help. This PDF file is provided so you can easily print multiple topics from the help information or read the online help in PDF format.

Note: Some topics do not convert properly to PDF, causing format problems. Some elements of online help are completely removed from the PDF version. Those problem topics can be successfully printed from within the online help.

Developing Extensions for HP OO

This document provides Java and .NET developers with guidelines for developing extensions for extending HP Operations Orchestration.

Note: Knowledge of Java or .NET is required.

You can extend HP Operations Orchestration programmatically. This means that third parties can add functionalities to HP Operations Orchestration and introduce them as content in the flow execution engine.

Introducing new content requires building an extension and deploying it to HP OO Central. You can write extensions in Java or .NET. This guide provides information on how to build extensions.

In HP Operations Orchestration 10.00, extensions are named plugins (in previous versions extensions were called IActions). A plugin is a piece of Java code running within the run engine. This piece of code can define its own isolated classpath. Classpath isolation ensures that different plugins can use conflicting dependencies. For example, plugin A can use dependency X version 1.0 and plugin B can use the same X dependency, but in version 2.0. You are now able to use both plugins in the same flow regardless of the conflicting classpath issue.

A plugin contains one or more actions and references to all required dependencies. An `@Action` is a method in a class. See the [Developing Extensions](#) chapter for additional information.

Although all the plugins are written in Java, HP Operation Orchestration also supports .NET actions. The actions written in .NET are referenced by a wrapping Java plugin. See the [.NET Extensions](#) chapter for additional information.

Note: In HP OO 10.00 there are annotated actions (`@Action`). The `IAction` interface from HP OO 9.00 is now deprecated. Users writing new content should refrain from implementing the `IAction` interface and instead write `@Action`'s.

Creating an @Action

The recommended way to build @Actions are as a Maven plugin.

You should use Apache Maven 3.0.3. or later to build your plugins. For details on how to build your plugins without using Maven, see ["Without using Maven" on page 1.](#)

Developing Plugins

This section describes how to develop plugins. A user can create a sample plugin using maven archetype, and use that this as a template for creating plugins.

Install Maven

Maven installed on a computer with the bin directory in the computers path. This enables you to run `mvn` from anywhere in the file system.

Create a Local Maven Repository

- Expand `sdk-dotnet-<version>.zip` and `sdk-java-<version>.zip` to:

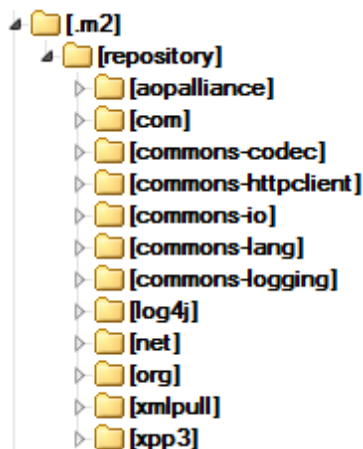
Windows: `%HOMEPATH%\m2\repository.`

Linux: `$HOME/.m2/repository.`

Note:

These files are located in on the ISO in the **SDK folder.**

- Following is an example of a directory structure, if the files were correctly extracted:



Register the Plugin Archetype

- Open the command prompt and enter the following command:

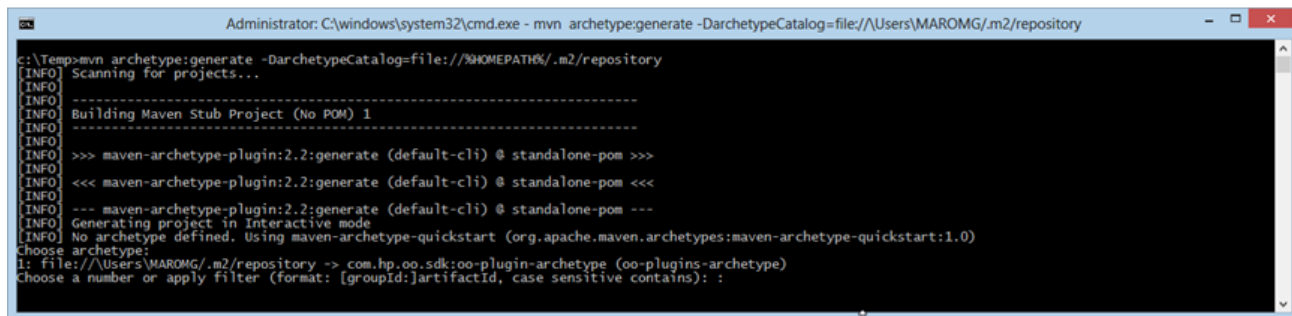
```
mvn archetype:crawl: This updates the maven archetype catalog under  
$HOME/.m2/repository.
```

Create a Sample Project

1. Go to the path where you want to create a sample plugin project and enter the following command:

```
mvn archetype:generate -DarchetypeCatalog=file://$HOME/.m2/repository (for  
Windows use %HOMEPATH%).
```

This initiates the project creation. First a list of archetypes found in the catalog appear, there should be only 1. Press **1** and then **Enter**.



```
Administrator: C:\windows\system32\cmd.exe - mvn archetype:generate -DarchetypeCatalog=file://\Users\MAROMG/.m2/repository  
c:\Temp>mvn archetype:generate -DarchetypeCatalog=file://%HOMEPATH%/.m2/repository  
[INFO] Scanning for projects...  
[INFO] -----  
[INFO] Building Maven Stub Project (No POM) 1  
[INFO] -----  
[INFO] >>> maven-archetype-plugin:2.2:generate (default-cli) @ standalone-pom >>>  
[INFO] <<< maven-archetype-plugin:2.2:generate (default-cli) @ standalone-pom <<<  
[INFO] --- maven-archetype-plugin:2.2:generate (default-cli) @ standalone-pom ---  
[INFO] Generating project in Interactive mode  
[INFO] No archetype defined. Using maven-archetype-quickstart (org.apache.maven.archetypes:maven-archetype-quickstart:1.0)  
Choose archetype:  
1: file://\Users\MAROMG/.m2/repository -> com.hp.oo.sdk:oo-plugin-archetype (oo-plugins-archetype)  
Choose a number or apply filter (format: [groupId]:artifactId, case sensitive contains):
```

2. While this is being created, enter the following:

- groupId: The group id for the resulting maven project, acmeGroup is used in the example above.
- artifactId: The artifact id for the resulting maven project, acmeArtifact is used in the example above.
- version: The version for the resulting maven project, 1.0 is used in the example above.
- package: The package for the files in the project. The default for this option is the same as the groupId.

3. After entering these values, when prompted press **Enter**:

```
Define value for property 'groupId': : acmeGroup  
Define value for property 'artifactId': : acmeArtifact  
Define value for property 'version': 1.0-SNAPSHOT: : 1.0  
Define value for property 'package': acmeGroup: :  
Confirm properties configuration:  
groupId: acmeGroup  
artifactId: acmeArtifact  
version: 1.0  
package: acmeGroup  
Y: :
```

4. The build will finish and a project is created.

```
[INFO] -----  
[INFO] BUILD SUCCESS  
[INFO] -----  
[INFO] Total time: 18.112s  
[INFO] Finished at: Mon Jun 17 21:17:50 IDT 2013  
[INFO] Final Memory: 13M/490M  
[INFO] -----  
c:\Temp>
```

Opening the Project

This created a new java project with a maven-based model. This project can be built, and the plugin-pack results can be imported into Studio. This project is located in the folder where the user initiated the `mvn archetype:generate` command under a folder that has the same name as the `artifactId` the user provided.

There are two modules inside the project:

- `plugin-example` module: Is an example for an OO plugin. It contains a single class with a single `@Action` inside. The result of building this module is a jar which is a maven plugin so it can be run independently from OO.
- `plugin-pack-example` module: Is an example for a plugin pack that packs one or more plugins inside. It is dependent on our `plugin-example` module, and it also uses a different plugin during its build to fetch all its dependencies so it will be able to run inside OO.

The product received by building this module is a jar that can be imported into Studio. After the import new operations can be created from the `@Actions` it contains.

Developing Extensions

As an extension is a method in a class, it can be any method in any class. This method is also referred to as an `@Action`.

An `@Action` is invoked during flow execution, when an operation using that `@Action` is executed.

"Hello World!" example

To mark a method as `@Action`, annotate it with `@com.hp.oo.sdk.content.annotations.Action`. The following is a simple "Hello World!" `@Action` example:

```
public class MyActions {
    @Action
    public void sayHello() {
        System.out.println("Hello World!");
    }
}
```

By default, the created `@Action` is named after the method that defines it. In the "Hello World!" example, the `@Action` name is `sayHello`. The `@Action` name is used in the operation definition. The operation is the mean to expose an `@Action` to Studio and to flow authors. Each operation points to a specific `groupId`, `artifactId`, `version` and `@Action` name (`GAV+@Action` name).

You can customize the `@Action` name and provide a name that is different from the method name. You can do this using the `@Action` annotation value parameter. The following code defines the same "Hello World!" `@Action`, but names it `my-hello-action`:

```
public class MyActions {
    @Action("my-hello-action")
    public void sayHello() {
        System.out.println("Hello World!");
    }
}
```

Passing Arguments to @Actions

An `@Action` is exposed to the flow context and can request parameters from it. The flow context holds the state of the flow. For example, consider the following `@Action`, which adds two numbers and prints the result to the console:

```
@Action
public void sum(int x, int y){
    System.out.println(x+y);
}
```

Parameters are taken from the context by name. The `sum` method requests two integer parameters `x` and `y` from the context. When invoking the `@Action`, HP Operations Orchestration assigns the value of `x` and `y` from the context to the method arguments having the same name.

Just like with `@Action`, it is possible to customize parameter names and request that HP Operations Orchestration resolves the value while using a custom name. In the following example,

the `sum` method requests that the context `op1` parameter is assigned to the `x` argument and `op2` to the `y` argument:

```
@Action
public void sum(@Param("op1") int x, @Param("op2") int y){
    System.out.println(x+y);
}
```

Return Values

An `@Action`, like any Java method, can also return a single value. The returned value is considered the return result of the `@Action` and is used as `return result` in the operation. It is also possible for an `@Action` to return multiple results to the operation. This is done by returning a `Map<String, String>`, where the `Map` key is the name of the result, and the associated value is the result value. Returning a `Map<String, String>` is a way for an `@Action` to pass multiple outputs to the operation at runtime.

Adding @Action Annotations

`@Action` annotations are used to generate new operations in the Studio. When generating an `@Action` based operation, the new operation's initial attributes (description, inputs, outputs, responses) are taken from the `@Action` annotations definitions.

Note: It is important that you use `@Action` annotations, otherwise operations created these `@Action`'s are harder to use.

Annotations

Adding metadata means adding or setting the relevant annotations and their attributes. The following table describes the `@Action`, `@Output`, `@Response` and `@Param` annotations:

Action

Attributes:

- `value` (optional): the name of the `@Action`
- `description` (optional)
- `Output[]` (optional): array of `Outputs` (see below)
- `Response[]` (optional): array of `Responses` (see below)

Comments:

You have two options for setting the name of the `@Action`:

1. The value attribute:

```
@Action("af1Ping")  
public void ping(...)
```

or

```
@Action(value="af1Ping")  
public void ping(...)
```

2. The method name:

```
@Action  
public void ping(...)
```

The names are checked in the above order. The first one checked is the value attribute. If it doesn't exist, the method name is selected.

Param

Attributes:

- value: the name of the input
- required (optional): by default is false
- encrypted (optional): by default is false
- description (optional)

Comments:

This is important not only for the @Action data, but also for execution.

Inputs give an operation or flow the data needed to act upon. Each input is mapped to a variable. You can create an input for a flow, operation, or step.

In Studio, inputs can be:

- Set to a specific value.
- Obtained from information gathered by another step.
- Entered by the person running the flow, at the start of the flow.

See the *HP OO 10 Studio Authoring Guide* for more information and the "[Passing Arguments to @Actions](#)" on page 11 for details on the execution functionality.

Output

Attributes:

- `value`: the name of the output
- `description` (optional)

Comments:

In order for the operation in studio to have multiple outputs, the `@Action` itself has to declare them. Assigning values to multiple outputs can be achieved by creating an `@Action` whose return value is a `Map<String, String>`.

The output is the data produced by an operation or flow. For example, success code, output string, error string, or failure message.

In Studio, the different kinds of operation outputs include:

- Raw result: the entire returned data (return code, data output, and error string).
- The primary and other outputs, which are portions of the raw result.

See the *HP OO 10 Studio Authoring Guide* for more information.

Response

Attributes:

- `text`: the text displayed by each response transition
- `field`: the field to evaluate
- `value`: the expected value in the `field`
- `description`: (optional)
- `isDefault`: Indicates whether this is the default response. The default value is `false`. Only one response in a `@Action` can have this set to `true`.
- `mathType` : The type of matcher to activate against the value. For example if we defined (`field = fieldName, value = 0, matchType = COMPARE_GREATER`) this means that this response will be chosen if the field `fieldName` will have a value greater than `0`.
- `responseType`: The type of the response (`Success, Failure, Diagnosed, No_Action` or `Resolve_By_Name`).
- `isOnFail`: Indicates whether this is the On-Fail response. The default value is `false`. Only one response in a `@Action` can have this set to `true`.
- `ruleDefined`: Indicates whether or not this response has a rule defined. Responses that have no rules defined can be used as the default response. There should be only one response without a rule defined in a single `@Action`.

Comments:

A response is the possible outcome of an operation or flow. The response contains a single rule: field matches value.

See the *HP OO 10 Studio Authoring Guide* for more information.

@Action Data Definition Example

```
@Action(value = "afIPing",
        description = "perform a dummy ping",
        outputs = {@Output(value = RETURN_RESULT, description = "returnResult description"),
                  @Output("packetsSent"),
                  @Output("packetsReceived"),
                  @Output("percentagePacketsLost"),
                  @Output("transmissionTimeMin"),
                  @Output("transmissionTimeMax"),
                  @Output("transmissionTimeAvg")},
        responses = {@Response(text = "success", field = RETURN_CODE, value = PASSED),
                    @Response(text = "failure", field = RETURN_CODE, value = FAILED)})

public Map<String, String> doPing(
    @Param(value = "targetHost",
           required = true,
           encrypted = false,
           description = "the host to ping") String targetHost,
    @Param("packetCount") String packetCount,
    @Param("packetSize") String packetSize) {
    ...
}
```

Testing Extensions

Testing Extensions as Part of the Project Build

As an @Action is a simple Java method, it is possible to test it using standard Java test tools such as JUnit, leveraging the normal lifecycle phases of a Maven project.

As the @Action itself is a regular method, it does not require invoking any HP Operations Orchestration components. The invocation can be a direct Java method invocation in the test case.

Testing Extensions Independently from the Command Line

Once packaged into a plugin, you can invoke extensions from the command line for test purposes. The following is an @Action example:

```
public class TestActions {
    @Action
    public int sum(@Param("op1") int x, @Param("op2") int y){
        return x+y;
    }
}
```

Suppose the TestActions class is in a plugin with the following groupId, artifactId and version (GAV): **com.mycompany:my-actions:1.0**

You can invoke the sum @Action from the command line as follows:

```
mvn com.mycompany:my-actions:1.0:execute -Daction=sum -Dop1=1 -Dop2=3 -X
```

The result of this command is a long trace. The -X option is required to see log messages. Towards the end of the trace you can see:

```
[DEBUG] Configuring mojo 'com.mycompany:my-actions:1.0::execute' with basic configurator -->
[DEBUG]   (f) actionName = sum
[DEBUG]   (f) session = org.apache.maven.execution.MavenSession@21cfa61c
[DEBUG] -- end configuration --
[DEBUG] Action result: action result = 4
```


.NET Extensions

In order to create content using .NET actions, you need to:

1. Create a DLL file containing the implementation of the desired @Actions, just like in version 9.x. The @Action class should implement an IAction interface.
2. Deploy the created DLL, including referenced libraries, to the local Maven repository, using `mvn install:install-file`. For more information on installing artifacts that were not built by Maven, see <http://maven.apache.org/plugins/maven-install-plugin/usage.html>
3. Generate an HP OO Maven plugin, wrapping the .NET action. To do this, you need to:
 - a. Create a **pom.xml** file. For POM references, see <http://maven.apache.org/pom.html>.
 - b. Under `<dependencies>`, add a list containing all the required DLLs. Define all DLL artifacts using `<type>dll</type>`.
 - c. Run the `mvn install` command from the folder containing the **pom.xml** file. This is considering that the Maven **bin** folder is contained in the system path.

The result is the Maven plugin, placed in the target folder and installed to the local Maven repository. The target folder location is relative to the current folder.

The content of the **pom.xml** is:

```
<?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-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>[my plugin groupId]</groupId>
  <artifactId>[my plugin artifactId]</artifactId>
  <version>[my plugin version]</version>

  <packaging>maven-plugin</packaging>

  <properties>
    <oo-sdk.version>[THE LATEST HP_OO_SDK_VERSION]</oo-sdk.version>
    <oo-dotnet.version>[THE LATEST HP_OO_DOTNET_VERSION]</oo-dotnet.version>
  </properties>

  <dependencies>
    <!-- required dependencies -->
    <dependency>
      <groupId>com.hp.oo</groupId>
      <artifactId>oo-dotnet-action-plugin</artifactId>
      <version>${oo-sdk.version}</version>
    </dependency>

    <dependency>
      <groupId>com.hp.oo</groupId>
      <artifactId>oo-dotnet-legacy-plugin</artifactId>
      <version>${oo-dotnet.version}</version>
    </dependency>
  </dependencies>
</project>
```

```

        <type>dll</type>
    </dependency>

    <dependency>
        <groupId>${project.groupId}</groupId>
        <artifactId>IAction</artifactId>
        <version>9.0</version>
        <type>dll</type>
    </dependency>
    <!-- end of required dependencies -->

    <dependency>
        <groupId>[groupId-1]</groupId>
        <artifactId>[artifactId-1]</artifactId>
        <version>[version-1]</version>
        <type>dll</type>
    </dependency>

    <dependency>
        <groupId>[groupId-2]</groupId>
        <artifactId>[artifactId-2]</artifactId>
        <version>[version-2]</version>
        <type>dll</type>
    </dependency>

    ...

    <dependency>
        <groupId>[groupId-n]</groupId>
        <artifactId>[artifactId-n]</artifactId>
        <version>[version-n]</version>
        <type>dll</type>
    </dependency>
</dependencies>

<build>
    <plugins>
        <plugin>
            <groupId>com.hp.oo</groupId>
            <artifactId>oo-action-plugin-maven-plugin</artifactId>
            <version>${oo-sdk.version}</version>
            <executions>
                <execution>
                    <id>generate plugin</id>
                    <phase>process-sources</phase>
                    <goals>
                        <goal>generate-dotnet-plugin</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>
</project>

```

In the following example:

- The POM file is named **example.pom.xml**.
- The **my-dotnet-actions.dll** contains the desired @Actions.
- The generated Maven plugin is **com.example:my-dotnet-plugin:1.0**.

```
<?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-instance"  
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.  
0.0.xsd">
```

```
  <modelVersion>4.0.0</modelVersion>  
  <groupId>com.example</groupId>  
  <artifactId>my-dotnet-plugin</artifactId>  
  <version>1.0</version>  
  <packaging>maven-plugin</packaging>  
  
  <properties>  
    <oo-sdk.version>2.190</oo-sdk.version>  
    <oo-dotnet.version>1.30</oo-dotnet.version>  
  </properties>  
  
  <dependencies>  
    <!-- required dependencies -->  
    <dependency>  
      <groupId>com.hp.oo</groupId>  
      <artifactId>oo-dotnet-action-plugin</artifactId>  
      <version>${oo-sdk.version}</version>  
    </dependency>  
    <dependency>  
      <groupId>com.hp.oo</groupId>  
      <artifactId>oo-dotnet-legacy-plugin</artifactId>  
      <version>${oo-dotnet.version}</version>  
      <type>dll</type>  
    </dependency>  
    <dependency>  
      <groupId>${project.groupId}</groupId>  
      <artifactId>IAction</artifactId>  
      <version>9.0</version>  
      <type>dll</type>  
    </dependency>  
    <!-- end of required dependencies -->  
    <dependency>  
      <groupId>com.example</groupId>  
      <artifactId>my-dotnet-actions</artifactId>  
      <version>1.0</version>  
      <type>dll</type>  
    </dependency>  
  </dependencies>  
  
  <build>  
    <plugins>  
      <plugin>  
        <groupId>com.hp.oo</groupId>  
        <artifactId>oo-action-plugin-maven-plugin</artifactId>  
        <version>${oo-sdk.version}</version>
```

```
    <executions>  
      <execution>  
        <id>generate plugin</id>  
        <phase>process-sources</phase>  
        <goals>  
          <goal>generate-dotnet-plugin</goal>  
        </goals>  
      </execution>  
    </executions>  
  </plugin>  
</plugins>  
</build>  
</project>
```

Legacy Actions

In order to create content using legacy actions, you need to:

1. Verify that you have a JAR containing the implementation of the desired actions, just like in version 9.x. The action class should implement an `IAction` interface.
2. Deploy the JAR, including referenced libraries, to the local Maven repository, using `mvn install:install-file`. For more information on installing artifacts that were not built by Maven, see <http://maven.apache.org/plugins/maven-install-plugin/usage.html>
3. Generate an HP OO Maven plugin, wrapping the legacy actions library. To do this, you need to:
 - a. Create a `pom.xml` file. For POM references, see <http://maven.apache.org/pom.html>.
 - b. Under `<dependencies>`, add a list containing all the required JARs.
 - c. Run the `mvn install` command from the folder containing the `pom.xml` file. This is considering that the Maven `bin` folder is contained in the system path.

The result is the Maven plugin, placed in the target folder and installed to the local Maven repository. The target folder location is relative to the current folder.

The content of the `pom.xml` is:

```
<?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-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>[my plugin groupId]</groupId>
  <artifactId>[my plugin artifactId]</artifactId>
  <version>[my plugin version]</version>

  <packaging>maven-plugin</packaging>

  <properties>
    <oo-sdk.version>[THE LATEST HP_OO_SDK_VERSION]</oo-sdk.version>
    <oo-dotnet.version>[THE LATEST HP_OO_DOTNET_VERSION]</oo-dotnet.version>
  </properties>

  <dependencies>
    <!-- required dependencies -->
    <dependency>
      <groupId>com.hp.oo</groupId>
      <artifactId>oo-legacy-action-plugin</artifactId>
      <version>${oo-sdk.version}</version>
    </dependency>
    <!-- end of required dependencies -->

    <dependency>
      <groupId>[groupId-1]</groupId>
      <artifactId>[artifactId-1]</artifactId>
```

```

        <version>[version-1]</version>
    </dependency>

    <dependency>
        <groupId>[groupId-2]</groupId>
        <artifactId>[artifactId-2]</artifactId>
        <version>[version-2]</version>
    </dependency>
...

    <dependency>
        <groupId>[groupId-n]</groupId>
        <artifactId>[artifactId-n]</artifactId>
        <version>[version-n]</version>
    </dependency>
</dependencies>

<build>
    <plugins>
        <plugin>
            <groupId>com.hp.oo</groupId>
            <artifactId>oo-action-plugin-maven-plugin</artifactId>
            <version>${oo-sdk.version}</version>
            <executions>
                <execution>
                    <id>generate plugin</id>
                    <phase>process-sources</phase>
                    <goals>
                        <goal>generate-legacy-plugin</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>
</project>

```

In the following example:

- The POM file is named **example.pom.xml**.
- The **my-legacy-actions.jar** contains the desired actions.
- The generated Maven plugin is **com.example:my-legacy-actions:1.0**.

```

<?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-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.example</groupId>
    <artifactId>my-legacy-actions-plugin</artifactId>
    <version>1.0</version>

    <packaging>maven-plugin</packaging>

```

```
<properties>
  <oo-sdk.version>2.190</oo-sdk.version>
  <oo-dotnet.version>1.30</oo-dotnet.version>
</properties>

<dependencies>
  <!-- required dependencies -->
  <dependency>
    <groupId>com.hp.oo</groupId>
    <artifactId>oo-legacy-action-plugin</artifactId>
    <version>${oo-sdk.version}</version>
  </dependency>
  <!-- end of required dependencies -->

  <dependency>
    <groupId>com.example</groupId>
    <artifactId>my-legacy-actions</artifactId>
    <version>1.0</version>
  </dependency>
</dependencies>

<build>
  <plugins>
    <plugin>
      <groupId>com.hp.oo</groupId>
      <artifactId>oo-action-plugin-maven-plugin</artifactId>
      <version>${oo-sdk.version}</version>
      <executions>
        <execution>
          <id>generate plugin</id>
          <phase>process-sources</phase>
          <goals>
            <goal>generate-legacy-plugin</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
</project>
```

