MyFaces Archetypes for Maven

{scrollbar}

MyFaces Maven archetypes

The MyFaces Maven archetypes project provides a number of Maven archetypes to get you started on a MyFaces-based project quickly. The table below lists the available archetypes and which MyFaces libraries are used for any given archetype.

align=left,align=left,align=center,align=center,align=center,align=left

Archetype name	Description of the created project	JSF version	View technology		Component set
			JSP	Facelets	
HelloWorld 2.0 Basic	Webapp project that uses MyFaces Core 2.0.x with Facelets as the page templating system.	2.0		•	none
HelloWorld 2.0 OpenWebBeans	Webapp project that uses MyFaces Core 2.0.x with Facelets as the page templating system. Uses JSR-299/JSR-330 annotations for managed bean definition with Apache OpenWebBeans as the CDI implementation.	2.0		•	none
HelloWorld Basic	Webapp project that uses MyFaces Core 1.2.x with JSP as the page templating system.	1.2	•		Tomahawk
HelloWorld-Facelets	Webapp project that uses MyFaces Core 1.2.x with Facelets as the page templating system.	1.2		•	Tomahawk
HelloWorld-Portlet	Simple portlet-enabled webapp.	1.2	•		none
Trinidad	Webapp that uses MyFaces Core 1.2.x and MyFaces Trinidad.	1.2	•		Trinidad 1.2
Trinidad 2.0 Archetype	Webapp that uses MyFaces Core 2.0.x and MyFaces Trinidad 2.0.	2.0	•		Trinidad 2.0
JSF-Component Library	Project that is intended to define new JSF components (rather than just use them). If you want to create a jar that defines a set of new custom JSF components then this archetype is a good starting point.	1.2	•		N/A
JSF-Component Library myfaces-archetype-jsfcomponents20	Project that is intended to define new JSF components (rather than just use them). If you want to create a jar that defines a set of new custom JSF components then this archetype is a good starting point.	2.0		•	N/A
myfaces-archetype-codi-jsf20	Hello World Archetype using MyFaces CODI	2.0		•	none
myfaces-archetype-codi-jsf12	Hello World Archetype using MyFaces CODI	1.2		•	none

Each project that is created based on one of these archetypes comes with JUnit and Log4j preconfigured. The projects also come with Jetty preconfigured, so you don't need a separate Java EE Web Container or Application Server to deploy the app to.

Getting started

To get started with a new project using one of the provided archetypes, we have to make sure Maven is installed on your system. Once Maven is installed, we can just type

mvn archetype:generate -DarchetypeCatalog=http://myfaces.apache.org

at a command line. Maven will present us a numbered list of the available archetypes. (This should resemble the list presented above.) After choosing a number and pressing Enter, Maven asks us for a group ID, artifact ID, version, and package. These are the default values used to create the project. After we confirm these settings, Maven creates the new, empty project for us.

The MyFaces Hello World Archetypes

This archetypes allows you to generate a template for a web application that uses MyFaces, based on the blank example application. So, creating the template of the application is as easy as executing:

mvn archetype:generate -DarchetypeCatalog=http://myfaces.apache.org

A list of available archetypes is show. Just select one of the 'Hello world' options. Answer the questions that follow to set the properties and you're all set! There are different 'Hello World' archetypes, see the table above for details.

That's all. The goal `archetype:create` downloads the proper artifacts from the main maven repo and creates the test project. Now you can start to work immediately.

Getting and installing the plugin locally

If you want to use the latest code of the archetype, the first thing you should do is to checkout the myfaces archetype source from the svn, using the command:

svn co http://svn.apache.org/repos/asf/myfaces/myfaces-build-tools/trunk/maven2-archetypes

Now, let's install the plugin locally (this supposes that you have maven 2.x already installed). Navigate to the plugin root folder and use the `mvn install` command. For example:

cd myfaces-archetype-helloworld mvn install Other archetypes
If you want to install any of the other archetypes locally, just replace `myfaces-archetype-helloworld` in the above command with the appropriate id for the archetype you want to install.

At this point, you have the plugin installed in your local system. You don't have to install it again unless you want to update it.

Other useful maven goals

There are some useful maven goals that are available for the generated archetypes. Suppose we have created a project based on one of the archetypes, with the following details: `groupId=myAppId` and `artifactId=testApp`. (`myAppId` and `testApp` are just examples, you can choose any name you like.)

If we want to test the basic application created, we can just run the command:

cd testApp mvn package

And we will have a war-file created at testApp/target folder!

This whole process can save a lot of time while setting up a new web application. And what is better, you have already the structure to use maven 🙂

How to switch versions (e.g. to 1.1.x)

Depending on the archetype we choose, a MyFaces 1.2.x or MyFaces 2.0.x project is created. Open the `pom.xml` file to see the versions if you're in doubt. Should you want to change the version used, you can simple edit the `pom.xml` file.

E.g. to setup a MyFaces 1.1.x project we have to change the MyFaces version numbers within the generated `pom.xml`. Example for version 1.1.5:

xml <dependency> <groupId>org.apache.myfaces.core</groupId> <artifactId>myfaces-api</artifactId> <version>1.1.5</version> <scope>compile< /scope> </dependency> <dependency> <groupId>org.apache.myfaces.core</groupId> <artifactId>myfaces-impl</artifactId> <version>1.1.5</version> <scope>compile</scope> </dependency>
Make sure the version of the **`-api` and** `-impl` libraries are the same!

Jetty plugin

The Jetty6 plugin has been added to the `pom.xml` of the archetype. This makes it easy to start your application, without the need to install and configure any container or application server. Just type

mvn -PjettyConfig clean jetty:run

This will create a war, launch the Jetty container and it will server your project at http://localhost:8080/testApp

The JSF Components Library Archetype

This archetype generates a maven multi-module project prepared for the development of custom JSF components. To create your new project, the only thing you have to do is to execute this command;

mvn archetype:generate -DarchetypeCatalog=http://myfaces.apache.org

It might be the case that the central archetype catalog is not yet up to date and does not contain the Component Library archetype. In that case, you have to install the archetype module locally, as described above under #Getting and installing the plugin locally. The archetype-id to use is `myfaces-archetype-jsfcomponents`.

Now, you can create the project in the folder of your choice! Just go to that folder and run the `archetype:create` goal.

cd yourFolder mvn archetype:create -DarchetypeGroupId=org.apache.myfaces.buildtools \ -DarchetypeArtifactId=myfaces-archetype-jsfcomponents \ -DarchetypeVersion=1.0.1 \ -DgroupId=myAppId \ -DartifactId=myJsfComponents

And there you have it, you can start coding without the hassle of project configuration. The archetype generates a multi-module project with this structure:

-+ project base (parent pom) | +--- core (component library) | +--- examples (examples for the components)

The layout of the project is almost identical to the layout of the MyFaces sandbox. This also ensures that your components could go to the sandbox without having too handle major conversions.

The base project

It contains the parent pom which lists the 'core' and 'examples' modules.

The component library (core project)

It will contain the sources of your components, as well as some tests (it should!). There is one demo component that comes with the archetype, so you can see how it is implemented and configured. This demo component works in every implementation (both MyFaces and Sun's RI)

The examples webapp

It is a webapp with the basic setup to start making demonstrations of your custom components without having to do special setups. There is already an example for the demo component. The home web page includes the version of the component library (the version of the pom file).
The webapp can be built with Myfaces or with the JSF-RI and can be executed directly with Jetty (the same way than the MyFaces Archetype). That is very handy to test that your components work in both implementations. For instance, if you want to execute your application directly from Jetty and using MyFaces, execute this in the examples folder:

mvn -PjettyConfig clean package jetty:run

Alternately, to build your example app using the JSF Reference Implementation, execute this in the examples folder:

mvn -PjettyConfig -Djsf=ri clean package jetty:run

And you can go to [http://localhost:8080/myJsfComponents-examples] to see the examples.

Always be sure to 'clean' when you switch JSF implementations, or your example app may contain jars from "both" MyFaces and the RI.

No other technologies are included (e.g. facelets), but it is easy to add whatever you need. There is no need to reinvent the wheel and following a standard structure will boost the code understanding between developers.

For more information on this archetype and how you can use it you can refer to this tutorial.

{scrollbar}