

# Tutorial: Simple Clearspace Action Plugin

You can build plugins that enhance Clearspace with links to new functionality, additions to the admin console, and additions to the end user UI.

This tutorial introduces you to the basics of building [action plugins](#). Action plugins incorporate WebWork actions, in which a user gesture (such as clicking a link) directs processing to an action class whose code executes, then typically routes processing back to the user interface to display some result.

The simple plugin you'll build with this tutorial displays a link that, when clicked, displays a new page showing a greeting. There are simpler ways to do this, of course, but hopefully you'll see how much more you can do in your action class and how useful it is to separate that code from your user interface.

**Note:** Other kinds of plugins include macros, filters, and web services. For more information, see [Overview: Building Macros and Filters](#), [Tutorial: Simple Clearspace Macro](#), and [Exposing Web Services Through a Plugin](#).

## Action Plugin Basics

In the model-view-controller (MVC) architecture that Clearspace is based on, the idea is to separate code that manages logic around data (the model) from code that presents the data as user interface (the view) from code that controls interactions between the two (the controller). Action plugins use the same model on a smaller scale. In this tutorial, you'll build a simple plugin that contains all of these pieces.

You'll create a plugin that adds a menu item to the Browse menu of the user bar (that row of menus along the top of every Clearspace page). That "Say Hello" menu item will simply navigate to a new page that displays a simple greeting. Here are the pieces you'll work with:

- A SimpleAction Java class that will do the work of returning the greeting to present.
- A simple-template.ftl FreeMarker template that will present the greeting.
- An xwork-plugin.xml file that will map the interaction, telling Clearspace to look at SimpleAction for the data needed and at simple-template.ftl for presenting it.
- The plugin.xml file, where you'll include descriptive info for this new functionality.

## What You'll Need to Get Started

This tutorial is about the basics, so you'll want only a few things to build all the pieces it describes:

- **Java 5.** Doesn't get more basic than that. If you don't have it, you'll find instructions for getting it in the readme of your...
- **Clearspace distribution** for testing; either Clearspace or ClearspaceX will work. Okay, that's pretty obvious, too. If you already have a Clearspace instance you can deploy to, then you're all set. (You'll also need to be able to log in to Clearspace as an admin.) This tutorial assumes you're using the standalone version (see the topic on setting up) because that's the simplest way to get started. You

can download the standalone version from [jivesoftware.com](http://jivesoftware.com). (Don't use a production deployment for a first go-round with new code – even simple rock solid code such as you'll write here!)

- **Apache Ant** for building and deploying your code. The tutorial makes heavy use of the included Ant build file to make compiling and deploying your plugin easier. You can get Ant at the [Apache web site](#).
- **An editor for code** – Java code and XML. IDEs such as Eclipse or IntelliJ IDEA are great, but more lightweight editors will do nicely, too. That's about it. Check out the next section for a few setup steps, then you can get started writing code.

## Setting Up

These setup steps are likely pretty common to other extension work you'll do – whether with macros or plugins or themes.

- Set up Clearspace. You can skip this if you've already got Clearspace installed and set up.
- Create spaces or communities to test in. You can skip this, too, if you've got spaces or communities to test in.
- Create a place to put your plugin project code. The Clearspace dev kit provides an Ant build file to make this a little quicker.

## Set Up Clearspace

If you haven't installed Clearspace and used its setup tool, use the following instructions. If you have, you can skip this part. Here are the steps:

1. Install your Clearspace development instance using the installation instructions.
2. Run Clearspace and use its setup tool to set basic configuration options, such as the location of the `jiveHome` directory. If you're unfamiliar with the setup tool, use the following setup tool settings:
  - For `jiveHome`, use `<distribution_root>/jiveHome`
  - For License, accept "Evaluation".
  - For Database Settings, choose "Evaluation Database".
  - Accept defaults for the rest.

## Set Up Your Project

It's easiest to develop plugins if you have your source and compiled artifacts in certain places. In particular, Clearspace expects your compiled Java classes to live in a `classes` directory under your plugin's root, your `plugin.xml` file will need to be at the root, and so on. Jive Software provides an Ant `build.xml` file you can use to create the project directory hierarchy, compile its sources, and deploy the result to your Clearspace instance for testing.

## If You're Using Another Distribution

This tutorial (and the Ant build file that goes with it) assumes you're using the Clearspace standalone distribution. Of course, you can use this tutorial with another Clearspace distribution you've set up for development and testing (such as a Clearspace WAR file deployed to your application server). You can

also set a `jiveHome` directory that's external to the distribution (this is the best practice for production). If you do either of these, you'll want to edit a couple of properties in the included Ant build file:

- Edit the `server.home.dir` property value to point to the root directory of your application server. Note that this is needed to find the `javax.servlet-api.jar` file, so you might need to edit the `<classpath>` element, too.
- Edit the `clearspace.lib.dir` property value to point to the directory that contains the libraries deployed with Clearspace. This might be at a path such as `<app_server_root>/webapps/clearspace/WEB-INF/lib`. The build file will resolve JAR file dependencies based on this path.
- Edit the `jiveHome` property to point to the `jiveHome` directory you specified when you ran the Clearspace setup tool. The build file will deploy your plugin oJAR file to a `plugins` directory inside `jiveHome`.

## Create a Project Hierarchy

1. Create a "SimpleExamples" directory for your plugin code. Get the Ant build file that's included with the dev kit, then copy it to the directory you created. If you're using the standalone Clearspace distribution with the included application server, set up your project directory at the following path inside the exploded distribution:

```
<distribution_root>/plugins/plugins/SimpleExamples
```

2. Grab the included Ant build file and copy it into the SimpleExamples directory you created..
3. Open a command prompt and navigate to the SimpleExamples directory.
4. Run the Ant target that creates the project directories:

```
ant create.plugin.dirs
```

Now that you've got a place for your code, you'll actually write some.

**Note:** At this point, if you don't know where your application log files are, you might want to locate them. The logs can be very helpful for debugging. If you're using the standalone distribution, you'll find the log files at `<dist_root>/server/logs`.

## Create a plugin.xml File

Every plugin has one. This file tells Clearspace what's in the plugin – in short, what Clearspace features you're extending – and where to find code and other supporting files your plugin needs (although not necessarily all of them, as you'll see in the next section).

1. Create a text file called `plugin.xml` in the root directory of your project (where the Ant build file is).
2. Paste the following into the new file:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<plugin xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://www.jivesoftware.com/schemas/clearspace/1_1/plugin.xsd"
  <name>SimpleExamples</name>
  <description>Simple macro and plugin examples.</description>
  <author>Me</author>
  <version>1.0.0</version>
```

```

<minServerVersion>1.3</minServerVersion>

<!-- The <components> element contains additions or customizations to
components in the user interface. Each <component> child element
represents a different UI piece, with its id attribute value
specifying which piece is being customized.

Here, you're customizing the user bar (the menu bar at the top of
each Clearspace page) so that its Browse menu gets a new entry
item, "Say Hello". The <url> element here specifies the URL that
Clearspace will load when the user clicks the item. In this
case, Clearspace will execute the sayhello action (which is defined
in the xwork-plugin.xml file).
-->
<components>
  <component id="user-bar">
    <tab id="jiveUserMenu5">
      <item id="sayhello" name="Say Hello">
        <url><![CDATA[<@ww.url action="sayhello" />]]></url>
      </item>
    </tab>
  </component>
</components>
</plugin>

```

Pretty simple, really. The `<plugin>` element's children include information about where the plugin is coming from (you), its version (in case you revise it for upgrade), and so on. The `<minServerVersion>` element specifies the minimum Clearspace version that the plugin will run on (Clearspace won't actually deploy it on earlier versions). The code here tells Clearspace to add a new "Say Hello" link to the Browse menu on the user bar. It also says which action (as defined in the xwork file you'll create in a moment) should be executed when the user clicks the link.

The plugin.xml can contain information about multiple bits of plugin functionality. For another kind of plugin, see Tutorial: Simple Clearspace Macro.

## Create a Java Class for the Action's Logic

Here, you'll create a Java action class that's your plugin's "model." This is where the plugin gets the data it needs, a simple greeting. A FreeMarker template you create in a moment will display the data.

1. In the src directory created by the create.plugin.dirs Ant target you ran, create the following Java package: com.example.clearspace.plugin.action.
2. In the new package, create a Java class called SimpleAction.java and paste the following code into it:

```

package com.example.clearspace.plugin.action;

import com.jivesoftware.community.action.JiveActionSupport;

/**
 * A "Hello World" plugin that merely receives or returns a greeting.
 * The JavaBeans-style accessors in this class are mapped to
 * property names in resources/simple-template.ftl, which is the FreeMarker
 * template that provides UI for displaying the data. The mapping is
 * done by WebWork after the class-to-FTL mapping in the
 * xwork-plugin.xml file.
 *
 * In other words, this class represents the plugin's data "model,"
 * the FTL file provides its "view," and the code in the xwork-plugin.xml
 * file provides its "controller."
 */
public class SimpleAction extends JiveActionSupport {

```

```

private static final long serialVersionUID = 1L;

private String message = "Hello World";

/**
 * Gets the greeting message. This method is mapped by WebWork to the
 * $message property used in simple-template.ftl. In other words,
 * in rendering the user interface, Clearspace (via WebWork) maps
 * the $message property to a "getter" name of the form
 * get<property_name> -- this getMessage method.
 *
 * @return The greeting text.
 */
public String getMessage() {
    return message;
}

/**
 * Sets the greeting message. The FTL file doesn't provide a way
 * for the user to set the message text. But if it did, this is
 * the method that would be called.
 *
 * @param message The greeting text.
 */
public void setMessage(String message) {
    this.message = message;
}
}

```

**Note:** If you're using an IDE, you'll want the following JAR files on your classpath to support code in this class (all should be located in the WEB-INF/lib directory of your Clearspace instance):

- clearspace-<version>.jar
- rife-continuations.jar
- webwork.jar
- xwork.jar

## Create a FreeMarker Template to Provide UI for the Action's Results

Now it's time to create the "view" for this part of your plugin. One important thing to notice is that the plugin class you just created and the view code you're about to create are in a way connected by a naming convention. In other words, the

```
getMessage
```

method in the class is matched up with the message variable in the code below through a convention in which Clearspace knows that removing the method name's "get" and lower-casing the first letter "m" creates a match with the variable. (Although, actually WebWork does all that behind the scenes.) You'll enable that mapping through the xwork file you'll create in a moment.

- In the resources directory created by the create.plugin.dirs Ant target you ran, create a file called simple-template.ftl and paste into it the following code:

```

<html>
  <head>
    <!-- Create a FreeMarker variable for the page title bar text, then
         use that variable in the <title> element. -->
    <#assign pageTitle="Hello World" />

```

```

        <title>${pageTitle}</title>
        <content tag="pagetitle">${pageTitle}</content>
    </head>
    <body>
        <!-- Have the message appear a little down on the page and
            in the center, so it's easier to find. -->
        <br/>
        <p align="center">${message}</p>
    </body>
</html>

```

## Create an xwork File to Map the Action to the Logic Class

This is where you'll connect the pieces. The code below defines an action that is associated with the SimpleAction action class you created. A "success" result returned by the class (something it does by default in this case) tells Clearspace to go get the simple-template.ftl template and process it by merging in the data that it retrieved from the class.

- At the root of the project hierarchy, where build.xml file is, create a file called xwork-plugin.xml and paste in the following code:

```

<!DOCTYPE xwork PUBLIC "-//OpenSymphony Group//XWork 1.1.1//EN"
"http://www.opensymphony.com/xwork/xwork-1.1.1.dtd">
<xwork>
    <package name="example-actions" namespace="" extends="community-actions">
        <!-- Map the action name, sayhello, to the action class, SimpleAction. -->
        <action name="sayhello" class="com.example.clearspace.plugin.action.SimpleAction">
            <!-- Specify the FTL file that should be used to present the data in the case of
                a "success" result (the default for an action class). -->
            <result
name="success">/plugins/simpleexamples/resources/simple-template.ftl</result>
        </action>
    </package>
</xwork>

```

## Build, Deploy and Test the Plugin

Use the build.plugins Ant target to compile and package the code into a JAR file, then use the deploy.plugins target to copy the plugin into the <jiveHome>/plugins directory.

1. At your command prompt, run

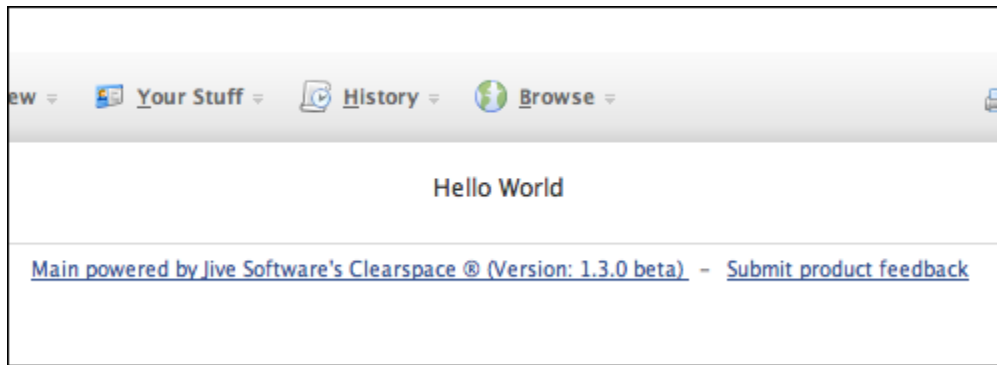
```
ant build.plugins
```

to build, then

```
ant deploy.plugins
```

to deploy to your server. There's no need to restart your server, but you'll want to wait five or ten seconds for Clearspace to deploy the plugin for use.

2. Open your browser to Clearspace.
3. Click the Browse menu, then click the Say Hello item.
4. Notice that you're taken to a new page that displays something like the following near the top:



That's it for this introduction to action plugins. As you can imagine, your action class could do much more – retrieve data from an external source (or from the Clearspace database using its API), perform calculations on data entered by the user, and so on. You could have multiple FTL files to provide different user interface responses to your plugin's state, such as what is returned by your action class.

To learn more about plugins, be sure to check out [Jivespace](#), Jive's developer community site.