

Clearspace Plugin Development Guide

Clearspace is a flexible and pluggable application. This document describes one aspect of customization, the plugin framework. Developers can use the plugin framework to execute arbitrary Java code and call the Clearspace APIs, as well as extends or override UI functionality.

This document includes:

- [System Requirements](#)
- [Plugin Structure](#)
- [Creating a New Plugin](#)
- [Creating Views with Plugins](#)
- [Database Access](#)

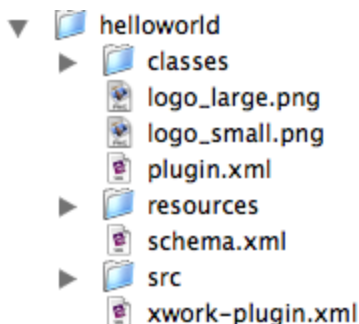
System Requirements

To write Clearspace plugins you'll need the following:

- A source build of Clearspace. This is available from your customer account on the [Jive Software](#) website. Any customer (licensee) of Clearspace is entitled to the source build and it is a standard part of the distribution.
- The Java 5 SDK. This is available on many platforms, mainly from the main Java site or from other vendors like Apple.
- (Optional) A Java IDE. Jive Software recommends development environments such as [JetBrain's IntelliJ](#), [Eclipse](#) or [TextMate](#). (Of course, plain text editors like [vi](#) or [Emacs](#) will work just fine as well.)

Plugin Structure

It's important to understand the parts of a plugin. The following lists the contents of an example plugin:



- Plugin Name: In this case "helloworld" is the name of the plugin.
- classes: Java classes are compiled in to this directory.

- PNG images: The two images (logo_large.png, logo_small.png) are optional and act as the plugin icon for the Clearspace admin console.
- plugin.xml: Metadata file for the plugin. Describes main class, resources, mappings and other properties.
- resources: Contains web resources and templates.
- schema.xml: An optional database definition. This will automatically install any tables your plugin uses.
- src: All Java code for the plugin.
- xwork-plugin.xml: A mapping file for any Webwork actions you create.

Creating a New Plugin

In the source release, create a new directory under plugin directory named the plugin name. This will be the base directory under the plugin. All source code will go under the src directory, class files should be compiled to the "classes" directory. All plugins can be compiled using the build.xml inside clearspace/build/build.xml by using the "plugins" target. All plugins will be deposited to the clearspace/target/ directory.

Implement the Plugin Interface

The first step to creating a plugin is to implement the com.jivesoftware.base.plugin.Plugin interface. The interface has two methods to implement: initializePlugin and destroyPlugin.

```
public class HelloWorldPlugin implements Plugin {
    public void initializePlugin(PluginManager manager, File pluginDirectory) {
        Log.info("Hello World!");
    }
    public void destroyPlugin() {
        Log.info("Good bye cruel world!");
    }
}
```

Configure

The core plugin configuration is contained in a plugin.xml file. You should put this file in the root directory of the plugin. This XML file contains the plugin name, plugin class name, version information, and other metadata.

```
<plugin>
  <class>com.jivesoftware.helloworld.HelloWorldPlugin</class>
  <name>helloworld</name>
  <description>Hello World</description>
  <author>Jive Software</author>
  <version>1.0.0</version>
  <minServerVersion>1.0.0</minServerVersion>
  <databaseKey>helloworld</databaseKey>
  <databaseVersion>1</databaseVersion>
</plugin>
```

Deploy

Once the plugin.xml is configured, you can deploy the plugin. Use the clearspace/build/build.xml and its plugins task to build the JAR file under the clearspace/target/plugins directory. Next, copy this JAR file to your jiveHome/plugins directory. The plugin will be automatically deployed once it is placed inside the directory. You don't need to restart the application server.

Creating Views with Plugins

You can create new pages using WebWork and FreeMarker templates. Also, you can override existing pages by overriding both the WebWork namespace and action name.

Using WebWork

WebWork configuration can be done in the xwork-plugin.xml. To create an action for the admin tool, community-admin-default package; otherwise, extend the community-default package. All pages must be written in FreeMarker. URLs to pages must contain /plugins/<pluginname> in front of the URL. See the following example:

```
<xwork>
  <package name="helloworld" namespace="/admin" extends="community-admin-default">
    <action name="hello" class="com.jivesoftware.helloworld.HelloWorldAction">
      <result name="success"
type="freemarker">/plugins/helloworld/resources/helloworld.ftl</result>
    </action>
  </package>
</xwork>
```

For more information on WebWork, visit <http://www.opensymphony.com/webwork/>.

Using FreeMarker

FreeMarker pages should be placed in the directory resources, along with any other images, etc. When referencing images with pages you should add the prefix /plugins/<pluginname> to the resources and use the ww.url WebWork FreeMarker transform:

```
<html>
  <head>
    <#assign pageTitle="Hello World" />
    <title>${pageTitle}</title>
    <content tag="pagetitle">${pageTitle}</content>

    <content tag="pageID">settings-binarybody</content>
    <content tag="pagehelp">
      <h3>${pageTitle}</h3>
      <p>
        This is the hello world test plugin
      </p>
    </content>
  </head>
  <body>

    <p>
    ${message}</p>
```

```
</body>
</html>
```

The following Clearspace utilities are always available to FreeMarker plugin pages.

name	class	description
JiveGlobals	com.jivesoftware.community.JiveGlobals	Controls access to a number of global properties in the application.
JiveConstants	com.jivesoftware.community.JiveConstants	Gets constant values representing various objects in Clearspace.
LocaleUtils	com.jivesoftware.util.LocaleUtils	For retrieving and converting locale-specific strings and numbers.
StringUtils	com.jivesoftware.util.StringUtils	Utility class to perform common String manipulation algorithms.
DateUtils	com.jivesoftware.util.DateUtils	A formatter for dates --- uses a fast internal date formatter.
SkinUtils	com.jivesoftware.community.util.SkinUtils	A collection of utility methods for use in Clearspace skins.
Permissions	com.jivesoftware.community.Permissions	Represents a set of permissions that an entity has for an object in the system.
IMSettingsUtils	com.jivesoftware.community.xmpp.IMSettingsUtils	IMSettingsUtils utility methods for accessing XMPP configuration settings as well as other IM settings.
ActionUtils	com.jivesoftware.community.action.ActionUtils	Utility class for dealing with WebWork actions.
RSSActionSupport	com.jivesoftware.community.action.RSSActionSupport	RSSActionSupport RSS-related utilities.
ViewCountManager	com.jivesoftware.community.stats.ViewCountManager	ViewCountManager how many views have been made on certain jive object.
CommunityUtils	com.jivesoftware.community.util.CommunityUtils	Utilities for handling communities.
BlogUtils	com.jivesoftware.community.util.BlogUtils	Utilities for handling blogs.
WikiUtils	com.jivesoftware.community.util.WikiUtils	Utility class for exposing wiki filter settings.
DocumentPermHelper	com.jivesoftware.community.util.DocumentPermHelper	DocumentPermHelper determining document permissions.

Database Access

Schema Installation Scripts

Database schema installation is performed by using SQLGen. SQLGen uses an XML format for table descriptions. There should be a file called schema.xml placed in the plugin home. Its format is as follows:

```
<schema name="Hello World">
  <table name="HelloWorld" description="Hello World Table">
    <column name="hwID" type="bigint" nullable="false" description="Primary Key"/>
    <column name="name" type="varchar" size="255" nullable="false" description="The name"/>
    <index type="primary" name="hw_pk" column="hwID" />
  </table>
</schema>
```

The following databases are supported:

- Oracle
- Postgres
- Mysql
- Sqlserver
- Derby
- Informix
- Hsql
- Mkoi
- DB2

Acquiring Database Connections

Database connections can be acquired by accessing the static getConnection() method of the com.jivesoftware.base.database.ConnectionManager class. This method will return a new connection from the underlying database pool. The connection must be released properly after use:

```
Connection con = null;
PreparedStatement psmt = null;
try {
  con = ConnectionManager.getConnection();
  ....
}
finally {
  ConnectionManager.close(psmt, con);
}
```

Note, the ConnectionManager has a number of overloaded close(..) methods.