

Tutorial: Simple Clearspace Widget

In this tutorial you'll build a simple Hello World widget. Widgets are views for displaying data on the Overview tab of a Clearspace space or community. System and space administrators can "design" the overview layout by dragging widgets onto the overview design area, setting widget properties, and arranging widgets in a particular layout. The admin can save the design in an unpublished state for editing later, then publish the overview so that people using the space can take advantage of its customized space view.

Widgets included with Clearspace provide views of content in Clearspace or on the web, or simply display a message. You can write your own widgets to give views of other content, or even act as enhanced versions of the included widgets.

Here's what the Overview page design space looks like with a widget on the left (in preview mode) and the right (in edit mode).

The screenshot displays the 'Test: Customize Overview' interface. At the top, there are links for 'Add widgets' and 'Copy another space'. Below this, a message states: 'Add, move, and configure widgets on this page. To add a widget, drag one from below into the page.' A grid of widget options is shown, including 'Hello World Widget' and 'HTML'. At the bottom of the grid are buttons for 'Publish Layout', 'Save for Later', and 'Discard Changes'. The bottom section of the interface is split into two panels. The left panel, titled 'Hello World Widget', shows the widget in preview mode with the text 'Hello World! Welcome to the Test community!'. The right panel, titled 'HTML', shows the widget in edit mode with a configuration panel. The configuration panel includes a 'Custom Title' field with the value 'HTML' and a 'Your HTML' field containing the following code:

```
<h3>Heading 3</h3>
<p>A paragraph. Lorem ipsum dolor, etc, etc, etc.</p>
```

 Below the configuration panel are buttons for 'Save Properties' and 'Cancel'.

Under the covers, a basic widget merely pulls together content and returns HTML for Clearspace to display when the widget is rendered. It can provide properties through which its user can guide what and how content is displayed. Along the way the widget might use an FTL file for presentation, properties files for internationalization, and so on.

Note: For information on using widgets to design an overview, see [Designing Pages with Widgets](#),

included with the Clearspace documentation.

This tutorial introduces you to the basics of building widget plugins. The simple widget you'll build with this tutorial displays a Hello World greeting.

Note: Other kinds of plugins include macros, actions, filters, and web services. For more information, see [Overview: Building Macros and Filters](#).

Widget Basics

Your widget can be very basic — such as a single class. But the widget can also include other files to support your design goals.

A basic widget could include:

- A Java class for logic (required). Your basic widget (with just a Java class) could return the HTML that Clearspace should display to the user. The widget class needs to implement the Widget interface, but you'll likely find it easier to do this by extending the BaseWidget class.
- FTL files for presenting data (optional). You can use an FTL file to shape the data you present to the user, handle widget resizing, and so on.
- A properties file to provide strings used for widget property names and descriptions (optional).
- i18n keys (optional).
- A plugin.xml file, as with other plugins.

In this tutorial, you'll create a widget that displays a "Hello World" message. Here are the pieces you'll work with:

- A HelloWorldWidget Java class that will do the work of returning the greeting to present.
- A hello-world.ftl FreeMarker template that will present the greeting.
- A .properties file that contains strings for the widget's UI.
- 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 a system or space administrator.) 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. (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 a space or community to test in. You can skip this, too, if you've already done it.
- Create a place to put your 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 easier 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. (For the build file and samples, grab the latest [developer kit](#) at Jivespace).

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 JAR 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`.

Writing Widget Code

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:

```
<plugin xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://www.jivesoftware.com/schemas/clearspace/1_8/plugin.xsd">
  <name>Hello World Widget</name>
  <description>Simple example of a widget that says hello.</description>
```

```
<author>Jive Software</author>
<version>1.0.0</version>
<minServerVersion>1.8.0</minServerVersion>

<!-- Defines a custom widget for this plugin -->
<widget class="com.jivesoftware.clearspace.plugin.example.widget.HelloWorldWidget" />
</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 deploy it on earlier versions). The code here tells Clearspace that the plugin includes a widget, and which widget class to load.

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 Widget's Logic

Here, you'll create a Java class that provides logic for the widget. Your widget class will implement the `com.jivesoftware.community.widget.Widget` interface, usually by extending the `com.jivesoftware.community.widget.BaseWidget` class. As you implement the class, you will:

- Implement three methods that Clearspace calls to display widget essentials.
- Add support for new properties with accessors and the `@PropertyNames` annotation.
- Call helper methods as needed to process an FTL file, support localization, and so on.

Note: For a basic widget, your classpath should include the main Clearspace JAR file, which is named `clearspace-<version>.jar`.

Required Methods

When you extend the `BaseWidget` class, you'll need to implement three methods from the `Widget` interface that aren't implemented in `BaseWidget`:

- `String getTitle(WidgetContext)` — This corresponds to your widget's default title. Implement this to return the title that should appear at the top of your widget.
- `String getDescription(WidgetContext)` — Implement this to return the description that should appear when the user hovers over your widget in the "customize" mode list of widgets.
- `String render(WidgetContext, ContainerSize)` — Implement this to return the HTML your widget will display when it's published.

You'll see that each of these methods takes a `WidgetContext` instance as a parameter. The `WidgetContext` class includes methods through which you can get information about the context in which the widget instance is executing -- its containing community, its current user, request and response objects, and so on. In some cases, you'll simply be passing the instance to other Clearspace methods.

Exposing Widget Design-Time Properties

Each widget exposes properties that the page designer can use to customize how the widget displays. For the two default properties — title and description — you need only implement getters to return the values that Clearspace should display. For properties you add, you do the following:

- Implement accessors through which Clearspace can get and set property values.
- Add a class-level `@PropertyNames` annotation that lists each of the property names.
- Include a `.properties` file that lists the String values Clearspace should use in the widget's UI where the user sets properties. (See the `.properties` file example below.)

In the following example, the title ("Hello World Widget") and description ("Displays a 'Hello World' message.") are properties implemented through the required accessors `getTitle` and `getDescription`.

Another property, `greetUser`, is implemented in the widget class through its own annotation and accessors; the radio buttons here are automatically provided because the accessors get and set a boolean value. The `greetUser` property also provides title and description values through a properties file included in the widget.

See the Java and property file examples in this topic.



Setting FreeMarker Properties

If your FTL file uses property variables, you can add values to the FreeMarker context from within your widget class. When Clearspace applies your template, the values you set will be used in generating the output.

Note that Clearspace doesn't add the widget class itself to the FreeMarker context in the way it does with action classes. This means that a `${x}` property in the FreeMarker template won't result automatically in a call to a `getX()` method in your widget class.

Instead, you should override the `BaseWidget.loadProperties(WidgetContext, ContainerSize)` method to add in any FreeMarker variables you need. Your implementation should call the superclass's

implementation to retrieve the properties Map, then set additional properties by putting them into the map. In other words, you'd add the x variable to the FreeMarker context using your loadProperties method implementation.

See the widget class code below for an example.

Helper Methods

BaseWidget provides helper methods you might find useful. Here are a few:

- String applyFreemarkerTemplate(WidgetContext, Widget.ContainerSize, String) — Call this to process an FTL file (whose path is given as the String parameter) and get the HTML result.
- String getLocalizedString(String, WidgetContext) — Call this to retrieve the string corresponding to the key i18n key you pass as the first parameter. See the section of this topic on using i18n keys.
- Map loadProperties(WidgetContext, Widget.ContainerSize) — Override this to add property values to the FreeMarker context. See the widget class example in this topic.

Writing the Java Code

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

```
package com.jivesoftware.clearspace.plugin.example.widget;

import java.util.Map;

import com.jivesoftware.community.annotations.PropertyNames;
import com.jivesoftware.community.widget.BaseWidget;
import com.jivesoftware.community.widget.WidgetContext;

@PropertyNames("greetUser")
public class HelloWorldWidget extends BaseWidget {

    // FreeMarker template for rendering preview and published widget.
    private static final String FREEMARKER_FILE = "/plugins/example/resources/hello-world.ftl";

    // To hold the value of a custom property.
    private boolean greetUser = false;

    /**
     * Called by Clearspace to get the description that should be displayed for
     * the widget when the user hovers over it in the "customize" mode list of
     * widgets.
     *
     * @param widgetContext
     *      Context in which the widget instance is executing.
     */
    public String getDescription(WidgetContext widgetContext) {
        return "Displays a 'Hello World' message.";
    }

    /**
     * Called by Clearspace to get the widget's default title. The user will be
     * able to change this. If they do, their new title will be set with a call
     * to the final method BaseWidget.setCustomTitle.
     *
     * @param widgetContext
     *      Context in which the widget instance is executing.
     */
}
```

```

    */
    public String getTitle(WidgetContext widgetContext) {
        return "Hello World Widget";
    }

    /**
     * Called by Clearspace to get the value for the greetUser property.
     *
     * @return true if the user should be greeted; false to greet the world.
     */
    public boolean getGreetUser() {
        return greetUser;
    }

    /**
     * Called by Clearspace to set the value for the greetUser property.
     *
     * @param greetUser
     *         true to greet the user; false to greet the world.
     */
    public void setGreetUser(boolean greetUser) {
        this.greetUser = greetUser;
    }

    /**
     * Called by Clearspace to get the HTML used to display the widget when it's
     * previewed or published.
     *
     * @param widgetContext
     *         Context in which the widget instance is executing.
     * @param containerSize
     *         An enum constant representing the size of the widget
     *         instance's current container: LARGE or SMALL.
     */
    public String render(WidgetContext widgetContext,
        ContainerSize containerSize) {
        // Process the included FTL file to render the HTML for display.
        String result = applyFreemarkerTemplate(widgetContext, containerSize,
            FREEMARKER_FILE);
        return result;
    }

    /**
     * Called by Clearspace to get properties for use in your FTL file. These
     * will be added to the FreeMarker context.
     *
     * @param widgetContext
     *         Context in which the widget instance is executing.
     * @param containerSize
     *         An enum constant representing the size of the widget
     *         instance's current container: LARGE or SMALL.
     * @return A map of the properties and their values.
     */
    protected Map<String, Object> loadProperties(WidgetContext widgetContext,
        ContainerSize containerSize) {
        // First load existing properties.
        Map<String, Object> properties = super.loadProperties(widgetContext,
            containerSize);

        // Get the name of the community this instance is in, then add it as a
        // property.
        String communityName = widgetContext.getCommunity().getName();
        String userName = widgetContext.getUser().getName();
        properties.put("communityName", communityName);
        properties.put("userName", userName);
        properties.put("greetUser", greetUser);

        return properties;
    }
}

```

Writing the Properties File

Next, you'll need to create a `.properties` file that provides the strings used in the design-time user

interface. In the widget JAR, this file will be located at `<jar_root>/classes/beans/HelloWorldWidget.properties`.

1. In the src directory, create a text file called `HelloWorldWidget.properties`.
2. Into the new file paste the following text:

```
# Resource bundle for Hello World Widget

# Author of widget
author=Me

# Version of the widget
version=1.0

# Properties - Display name and description
greetUser.displayName=Greet the User
greetUser.shortDescription=Greets the user by name; otherwise, greets the World.
```

Notice that the "greetUser" part of the keys matches the name of the property as given in the `@PropertyNames` annotation.

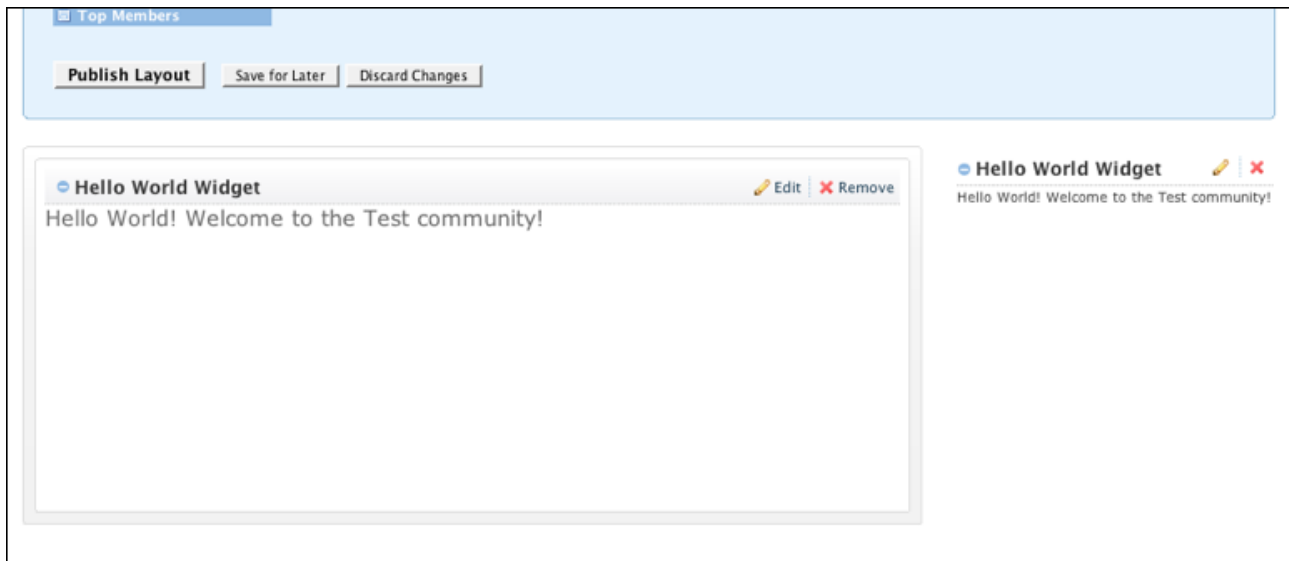
Writing an FTL File

You'll use a FreeMarker template (FTL) file to shape the data you're presenting in your published widget. The FTL file must be on the Clearspace classpath; that's easily accomplished by including it in your widget JAR file, then loading it at the relative location using the `applyFreemarkerTemplate` method (see the widget class code example in this topic).

Handling Widget Resizing

When someone clicks the "customize" link to begin designing a layout for a space's Overview tab, the design space includes a list of installed widgets and a three-column area for arranging widgets. Yet while it's a three-column area, a widget can only go on the left side (which is two columns wide) or the right side (which is one column wide). This gives a two-thirds/one-third look to the design. When someone drags your widget from the left side to the right side of the Overview tab design space, Clearspace resizes the widget from a large (two-column) size to a small (one-column) size.

Here's a view with the same widget on both sides:



Each widget is responsible for its display characteristics, including adjusting within its frame when its size changes. In your FTL file, you can handle each of these cases by providing a large rendering and a small one. You do that by testing for the value of the enum `com.jivesoftware.community.widget.Widget.ContainerSize`; its two values are `LARGE` and `SMALL`.

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

```
<style type="text/css">
  .jive-widget .jive-widget-body p.gobig {
    font: 16px verdana, helvetica, sans-serif;
  }
  .jive-widget .jive-widget-body p.getsmall {
    font: 10px verdana, helvetica, sans-serif;
  }
</style>

<!-- Use the widget's greetUser property to define the greeting style. -->
<#if greetUser>
  <#assign greeting = "Hello " + userName + "!">
<#else>
  <#assign greeting = "Hello World!">
</#if>

<!-- Render for display in a small area. -->
<#if containerSize == enums['com.jivesoftware.community.widget.Widget$ContainerSize'].SMALL>
  <p class="getsmall">${greeting} Welcome to the ${communityName} community!</p>
<#-- Render for display in a large area. -->
<#else>
  <p class="gobig">${greeting} Welcome to the ${communityName} community!</p>
</#if>
```

Build, Deploy and Test the Widget

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. Navigate to a space for which you're an administrator.
4. Click the Customize link, then click drag the Hello World Widget from the upper part of the page and drop it on the design space beneath.
5. Notice that the widget displays a greeting.
6. Click the edit link to view and edit the widget's properties.
7. Click Save Properties to save your changes, then click Publish to publish the overview with the new widget on it.

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