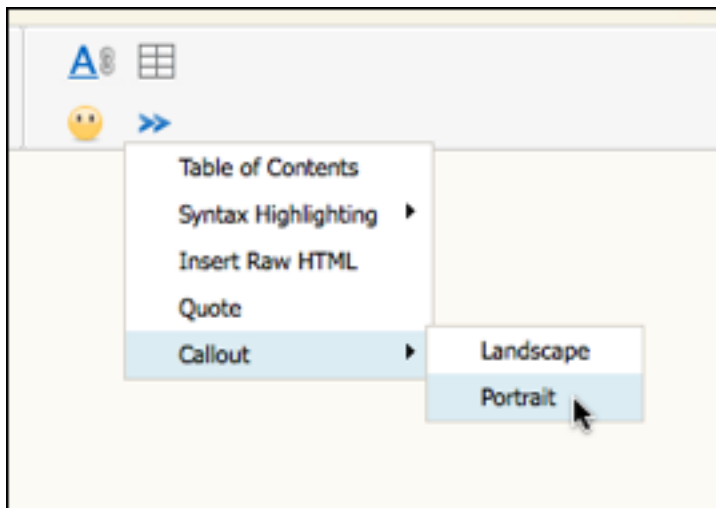


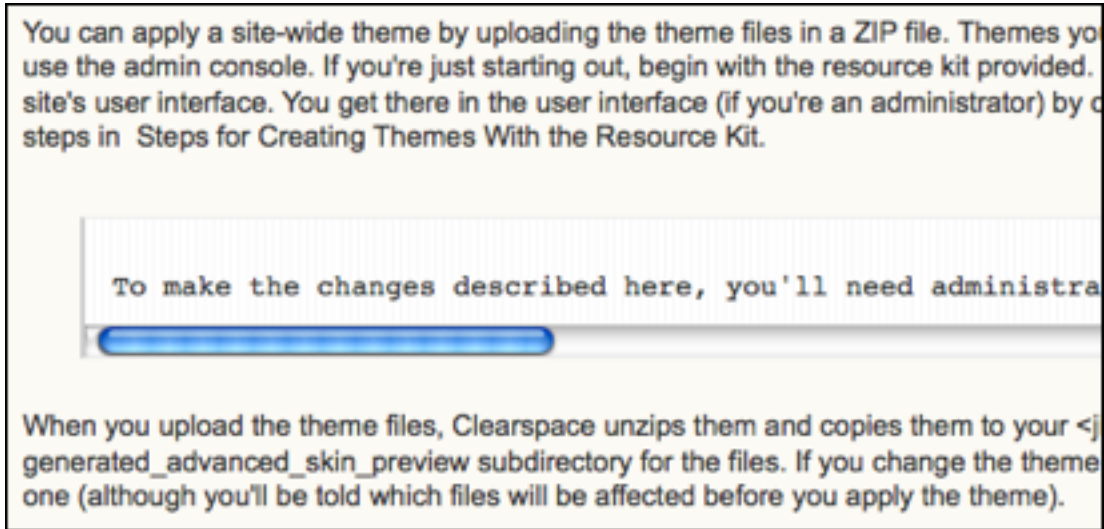
Building Macros

This introduction gives you the basics on building macros -- tools to make it easier for people to add and format content through the content editor. Using the macros included by default, people can insert a document table of contents, quote selected text, and format text as special syntax such as Java and XML.

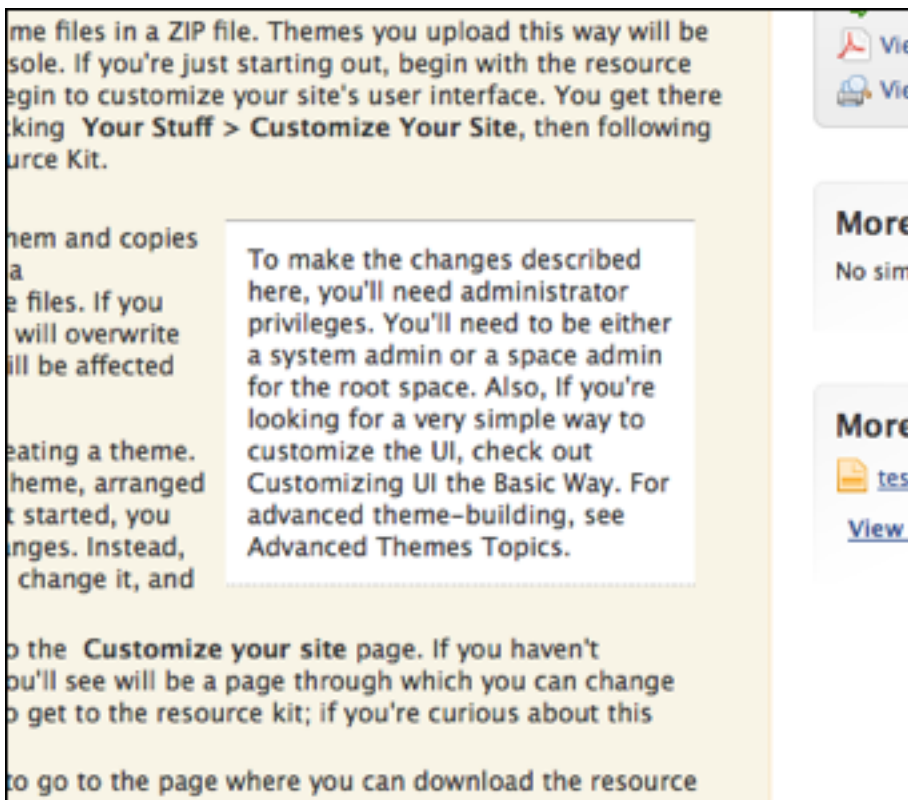
Note that the way people work with macros is different than it was in versions prior to 2.5. In earlier versions they used the plain text editor, adding macros by typing special tags in wiki markup. As of version 2.5, people add macros to content by using the Insert menu on the toolbar. The menu displays the macros that are installed in the application. A macro can also have suboptions, which you create in your macro by adding parameter sets.



The following illustrations shows the macro described in this topic, which inserts a callout (a box of text around which the main text flows). The first illustration shows the macro as it appears in the content while it's being edited.



This picture shows how the macro's published output is rendered.



Macro Basics

A very basic macro is made of:

- A plugin.xml file for configuration, as with other plugins (required).
- A Java class for logic (required). A macro class returns the HTML to render for the user when they publish content that includes the macro. The macro class needs to implement the com.jivesoftware.base.plugin.Macro interface.

Create a plugin.xml File for Configuration

As with other plugins, a macro needs a plugin.xml file for deployment. The `<macro>` element tells the application what it needs to know to deploy and present the macro. Notice in the comments below that the XML specifies how the macro should be displayed to the user -- for example, that it's displayed on the rich text editor's Insert menu and that it supports text typed by the user.

```
<plugin xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://www.jivesoftware.com/schemas/clearspace/2_5/plugin.xsd">
  <name>callout</name>
  <description>Provides support for inserting callouts (blocks of text
    around which primary content text flows)</description>
  <author>Jive Software</author>
  <version>1.0</version>
  <minServerVersion>2.5.3</minServerVersion>

  <!--
    The macro element includes basic information about the macro,
    as well as the parameter sets it supports, if any. Here, the
    callout macro is the kind that can contain text and will
    display on the insert menu of the content editor.

    - The name attribute's value is what will be displayed in the UI.
    - The type attribute specifies that this macro accepts text typed by
      the user.
    - The hasBody attribute specifies whether the macro accepts text
      typed by the user.
    - The showInRTE attribute specifies whether the macro should appear in
      the rich text editor's Insert menu.

  -->
  <macro name="Callout"
    type="text"
    hasBody="true"
    class="com.jivesoftware.plugin.CallOut"
    showInRTE="true">
  <!--
    A parameter set is a predefined set of values you use
    to configure your plugin. Parameter set names will appear
    in the content editor's insert menu. The macro's user
    chooses one of the names to insert the macro with the
    parameter values you assign here. Your macro class
    receives parameters of the selected parameter set in a
    HashMap.

  -->
```

```

<parameterSet name="Landscape">
  <parameter name="width" value="500px" />
  <parameter name="border-width" value="1px" />
  <parameter name="alignment" value="left" />
  <parameter name="background-color" value="white" />
</parameterSet>
<parameterSet name="Portrait">
  <parameter name="width" value="200px" />
  <parameter name="border-width" value="1px" />
  <parameter name="alignment" value="right" />
  <parameter name="background-color" value="white" />
</parameterSet>
</macro>
</plugin>

```

Defining and Handling Parameter Sets

A parameter set defines suboptions that a person sees when they select your macro from the Insert menu. For example, the Syntax Highlighting macro has four parameter sets: SQL, XML, Java, and Plain. The Callout macro has two parameter sets: Landscape and Portrait.

You define parameter sets in the plugin.xml. Each is a stanza with parameter elements as children. The parameter elements define name/value pairs that will be passed to your macro code when the application renders the content containing the macro. Here's an example from the Callout macro:

```

<parameterSet name="Landscape">
  <parameter name="width" value="500px" />
  <parameter name="border-width" value="1px" />
  <parameter name="alignment" value="left" />
  <parameter name="background-color" value="white" />
</parameterSet>
<parameterSet name="Portrait">
  <parameter name="width" value="200px" />
  <parameter name="border-width" value="1px" />
  <parameter name="alignment" value="right" />
  <parameter name="background-color" value="white" />
</parameterSet>

```

In other words, by selecting a single command from the Insert menu, the user is actually specifying multiple values. The list of parameters from the parameter set are passed to your macro's render method in a HashMap. The example below illustrates how to take the values and use them in code.

Create a Java Class for the Macro's Logic

You create a Java class that provides logic for the macro. The macro class implements the `com.jivesoftware.base.plugin.Macro` interface. The interface's `render` method should return the HTML that will be displayed when the content containing the macro is published.

The `render` method receives the following from the application:

- A text parameter containing the text, if any, that the person using the macro typed. In the Callout macro, that text is what should appear in the callout box.
- A parameter map containing the parameters your macro supports, mapped to their values. The Callout macro supports parameters that translate into CSS styles rules guiding how the callout is displayed: width, border-width, alignment, and background-color.
- A macro context object that contains information about the context in which the macro is being published.

Here's code for a simple macro class:

```
package com.jivesoftware.plugin;

import com.jivesoftware.base.plugin.Macro;
import com.jivesoftware.base.plugin.MacroContext;

import java.util.Map;

/**
 * A macro that provides support for inserting callouts (blocks of text around
 * which primary content text flows).
 */
public class CallOut implements Macro
{
    /**
     * Renders the macro's published output. Takes text the user entered and
     * wraps it in a text block. The block will be either narrow and at the
     * right side of the page ("portrait"), or wide and at the left side of the
     * page ("landscape").
     *
     * @param text
     *     The text that the user typed and applied the macro to.
     * @param paramMap
     *     Parameter name/value pairs selected by the user as a parameter
     *     set.
     * @param macroContext
     */
}
```

```

*      Context information about the macro's run time use, including
*      its user, containing plugin, containing content, and so on.
* @return The HTML to use in the published content.
*/
public String render(String text, Map<String, String> paramMap,
    MacroContext macroContext)
{
    // Get the parameter values chosen by the user in parameter sets.
    // Parameters given in parameter sets are defined in the macro's
    // plugin.xml file.
    String width = paramMap.get("width");
    String border = paramMap.get("border");
    String alignment = paramMap.get("alignment");
    String backColor = paramMap.get("background-color");

    // Build the return value, an HTML tag that-width opens the callout block.
    String markup = "<p style = 'padding: 10px; "
        + "margin: 10px 15px 10px 0px; "
        + "border-color: #b3b3b3; "
        + "border-top-style: solid; "
        + "border-bottom-style: dotted; "

        // Assemble the styles specified by the parameter set.
        + "border-width: " + border + " 0 " + border + " 0; "
        + "float: " + alignment + "; "
        + "background-color: " + backColor + "; "
        + "width: " + width + ">";

    // Start building the return value.
    StringBuilder calloutBox = new StringBuilder();
    calloutBox.append(markup);

    // Append the text the user typed.
    calloutBox.append(text);

    // Append the closing tag.
    calloutBox.append("</p>");

    return calloutBox.toString();
}
}

```