

Developing Macros

You use macros to modify content within a particular document, discussion or blog post. A user adds a macro to content to achieve a particular effect on text, embed external content, and so on. When content that includes the macro is published, Clearspace replaces the macro with HTML markup it retrieves from your macro's code.

For example, a macro could be used to replace comma-separated content with a table:

```
{csvtable}
name,age
Andrew,28
Nick,25
Pete,36
Bill,29
Aaron,31
Bruce,32
{csvtable}
```

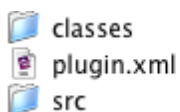
would render as:

name	age
Andrew	28
Nick	25
Pete	36
Bill	29
Aaron	31
Bruce	32

Note: For a sample of this, see the the [CSVTable Macro](#) at Jivespace.

Directory Structure

When you create a macro, you create a plugin with (typically) a smaller version of the same directory structure and layout described in the [Clearspace plugin development guide](#). The following illustration shows the directory hierarchy that's best for developing your macro plugin.



Java source files will be stored in the src directory, with their compiled results in the classes directory. The plugin.xml file tells Clearspace what's included in the plugin, where to find related files, and so on. You can also include a lib directory for third-party JAR files your code needs.

Implementing a Macro Class

A simple macro really just has two required parts: a plugin.xml file (every plugin has one) and a macro class that provides the markup Clearspace displays for the macro when it's published.

A macro class implements `com.jivesoftware.base.plugin.Macro`, an interface with a single method. Clearspace calls your implementation of the interface's render method to get the HTML you'd like to have displayed in place of the macro when content containing it is published.

Here's a brief example (imports and implementation have been removed to keep it short). This macro just takes the body (the text between the macro's tags), formats it and returns it. But the render method's two other parameters can be very useful:

- In the `parameters` parameter, Clearspace includes a map of parameters, if any, set by the user. Here's a macro with parameters: `{mymacro:id=foo|description=bar}`. The parameter names ("id" and "description") would be map keys.
- The `macroContext` parameter contains a wealth of useful information about the context in which your macro is being executed. Check out the Macro interface's Javadoc for more information.

```
public class CSVMacro implements Macro {
    /**
     * Called by Clearspace to retrieve HTML markup for the macro.
     *
     * @body The text between the macro tags, if any.
     * @param parameters Macro parameters set by the user, if any.
     * @param macroContext Information about the macro's context.
     */
    public String render(String body, Map<String, String> parameters, MacroContext
macroContext) {
        return csvToWikiString(body);
    }

    /**
     * Converts comma-separated text to a wiki table.
     *
     * @param macroBody The text between the macro tags that
     * should be formatted in a table.
     */
    private String csvToWikiString(String macroBody) {
        // Code omitted to keep this short. See the csv sample for the implementation.
    }
}
```

Configuration With a plugin.xml File

Here's a very brief plugin.xml snippet that defines a macro:

```
<plugin>
  <name>macroexample</name>
  <description>Example of making a macro.</description>
  <author>Me</author>
  <version>1.0.0</version>
  <!--
    The earliest Clearspace version that supports this macro. Note that
    the macro element below and the Macro interface your macro class implements
    are supported on versions 1.4 and later. See documentation included with
    earlier versions of Clearspace for information on those versions.
  -->
```

```
<minServerVersion>1.4.0</minServerVersion>

<!--
  The macro element specifies that the plugin contains a macro. The name
  attribute gives the macro's tag name, the hasBody attribute tells whether
  the macro is used as two tags and body in between ("true") or a single
  tag with no body ("false"), and the class attribute indicates which
  Java class (included in your plugin JAR file) implements your macro's logic.
-->
<macro name="csvtable" hasBody="true"
class="com.jivesoftware.clearspace.plugins.csv.CSVMacro" />
</plugin>
```