WebDAVHTMLArea

HTMLArea

This tutorial shows you how to extend the "step4" sample of Cocoon's WebDAV block with the HTMLArea control for WYSIWYG editing.

Copu the complete "step4" folder to a new "step6" folder.

Get the version 3.0rc1 (or later) of HTMLArea. See instructions on the HTMLArea sourceforge site.

Within the "step6" folder create another folder named "editor" and copy the contents of the "htmlarea" folder you just checked out into the "editor" folder.

Add the following entry to the "sitemap.xmap" file in the "step6" folder (after the "write/**" match is a good place):

```
<map:match pattern="editor/**">
  <map:read src="{global:staging}editor/{1}"/>
</map:match>
```

Then open your "file2html.xsl" stylesheet (in the step6/styles" directory) and change the template matching the root element to look like this:

```
<xsl:template match="/page">
  <html>
    <head>
      <!-- Configure the path to the editor.  We make it relative now, so that the
      example ZIP file will work anywhere, but please NOTE THAT it's better to
      have it an absolute path, such as '/htmlarea/'. -->
      <script type="text/javascript">
        _editor_url = "<xsl:value-of select="substring-before($requestURI, $sitemapURI)"/>editor";
        _editor_lang = "en";
      </script>

      <!-- load the main HTMLArea file, this will take care of loading the CSS and
      other required core scripts. -->
      <script type="text/javascript">
        <xsl:attribute name="src"><xsl:value-of select="substring-before($requestURI, $sitemapURI)"/>editor
/htmlarea.js</xsl:attribute>
      </script>

      <!-- load the plugins -->
      <script type="text/javascript">
        // WARNING: using this interface to load plugin
        // will _NOT_ work if plugins do not have the language
        // loaded by HTMLArea.

        // In other words, this function generates SCRIPT tags
        // that load the plugin and the language file, based on the
        // global variable HTMLArea.I18N.lang (defined in the lang file,
        // in our case "lang/en.js" loaded above).

        // If this lang file is not found the plugin will fail to
        // load correctly and NOTHING WILL WORK.

        //HTMLArea.loadPlugin("TableOperations");
        //HTMLArea.loadPlugin("SpellChecker");
        //HTMLArea.loadPlugin("FullPage");
        //HTMLArea.loadPlugin("CSS");
        //HTMLArea.loadPlugin("ContextMenu");
        //HTMLArea.loadPlugin("HtmlTidy");
        //HTMLArea.loadPlugin("ListType");
        //HTMLArea.loadPlugin("CharacterMap");
        //HTMLArea.loadPlugin("DynamicCSS");
      </script>

      <script type="text/javascript">
      var editor = null;

      function initEditor() {

        // create an editor for the "para" textbox
```

```
  editor = new HTMLArea("para");

  // register the FullPage plugin
  //editor.registerPlugin(FullPage);

  // register the SpellChecker plugin
  //editor.registerPlugin(TableOperations);

  // register the SpellChecker plugin
  //editor.registerPlugin(SpellChecker);

  // register the HtmlTidy plugin
  //editor.registerPlugin(HtmlTidy);

  // register the ListType plugin
  //editor.registerPlugin(ListType);

  //editor.registerPlugin(CharacterMap);
  //editor.registerPlugin(DynamicCSS);

  // register the CSS plugin
  /*editor.registerPlugin(CSS, {
    combos : [
      { label: "Syntax:",
                  // menu text       // CSS class
        options: { "None"           : "",
                   "Code" : "code",
                   "String" : "string",
                   "Comment" : "comment",
                   "Variable name" : "variable-name",
                   "Type" : "type",
                   "Reference" : "reference",
                   "Preprocessor" : "preprocessor",
                   "Keyword" : "keyword",
                   "Function name" : "function-name",
                   "Html tag" : "html-tag",
                   "Html italic" : "html-helper-italic",
                   "Warning" : "warning",
                   "Html bold" : "html-helper-bold"
                 },
        context: "pre"
      },
      { label: "Info:",
        options: { "None"           : "",
                   "Quote"          : "quote",
                   "Highlight"      : "highlight",
                   "Deprecated"     : "deprecated"
                 }
      }
    ]
  });
*/
  // add a contextual menu
  //editor.registerPlugin("ContextMenu");

  // load the stylesheet used by our CSS plugin configuration
  //editor.config.pageStyle = "@import url(custom.css);";

  editor.generate();
  return false;
}

HTMLArea.onload = initEditor;

function insertHTML() {
  var html = prompt("Enter some HTML code here");
  if (html) {
    editor.insertHTML(html);
  }
}
function highlight() {
```

```
        editor.surroundHTML('<span style="background-color: yellow">', '</span>');
      }
      </script>
    </head>
    <body onload="initEditor()">
      <form method="get">
        <xsl:attribute name="action"><xsl:value-of select="substring-before($requestURI, $sitemapURI)"/>write
/<xsl:value-of select="$file"/></xsl:attribute>
        <p>Title:<br />
        <input name="title" type="text" size="30" maxlength="30" value="{page/title}" />
        </p>
        <xsl:apply-templates select="metapage"/>
        <xsl:apply-templates select="page/content/para"/>
        <input type="submit" value="Submit" />
        <input type="reset" value="Reset" />
      </form>
    </body>
  </html>
</xsl:template>
```

Change your textarea to have to have 100 cols and 30 rows and give it an id of "para":

```
<xsl:template match="page/content/para">
  <p>para:<br />
    <textarea id="para" name="para" cols="120" rows="30">
      <xsl:copy-of select="node()"/>
    </textarea>
  </p>
</xsl:template>
```

Change all files in the "repo" directory (and its subdirectories) to contain only a single "para" element.

You can now view your work by visiting http://127.0.0.1:8888/samples/blocks/webdav/step6/repo/. This works quite nicely, however there is a flaw. If you look at your edited files the generated HTML is escaped.

The RequestGenerator has a special feature that allows you to put XML in request parameters and this XML gets fed into the pipeline as SAX events and gets processed by the SourceWritingTransformer appropriately. All you have to do is to give your request parameter a name that starts with "xml:". However this does not work in our case as HTMLArea does not generate valid XML but HTML.

The solution at hand is to use the HTMLTransformer. Declare it at the beginning of the sitemap, in the "components" element, right after the TraversableGenerator:

```
  <map:transformers default="xalan">
    <map:transformer name="html" src="org.apache.cocoon.transformation.HTMLTransformer">
      <jtidy-config>jtidy-config.txt</jtidy-config>
    </map:transformer>
  </map:transformers>
```

"jtidy-config.txt" has to be created in the "step6" directory with the following content:

```
output-xhtml: no
tidy-mark: no
```

This is necessary since the HTMLTransformer defaults to output XHTML, but HTMLArea does not work if the loaded HTML contains the XHTML namespace declaration.

Modify the writing pipeline to contain the HTMLTransformer:

```
<map:match pattern="write/**">
  <map:generate type="request"/>
  <map:transform src="{global:staging}styles/request2doc.xsl"/>
  <map:transform type="html">
    <map:parameter name="tags" value="para"/>
  </map:transform>
  <map:transform src="{global:staging}styles/doc2write.xsl">
    <map:parameter name="file" value="{global:staging}{1}"/>
  </map:transform>
  <map:transform type="write-source"/>
  <map:serialize type="xml"/>
</map:match>
```

As a last step you have to modify the "doc2write.xsl" stylesheet

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:req="http://apache.org/cocoon/request/2.0"
                xmlns:source="http://apache.org/cocoon/source/1.0">
<xsl:param name="file"></xsl:param>
<xsl:template match="request/parameters">
<page>
  <source:write create="true">
    <source:source><xsl:value-of select="$file"/></source:source>
    <source:path>page</source:path>
    <source:fragment>
      <title><xsl:value-of select="title"/></title>
      <content>
        <xsl:for-each select="content/para">
        <para>
          <!--the following line is the one to change-->
          <xsl:copy-of select="html/body/*"/>
        </para>
        </xsl:for-each>
      </content>
    </source:fragment>
  </source:write>
  <source:write create="true">
    <source:source><xsl:value-of select="$file"/>.meta</source:source>
    <source:path>metapage</source:path>
    <source:fragment>
      <author><xsl:value-of select="author"/></author>
      <category><xsl:value-of select="category"/></category>
      <state><xsl:value-of select="state"/></state>
    </source:fragment>
  </source:write>
</page>
</xsl:template>
</xsl:stylesheet>
```

To see how to more closely control formatting freedom of the authors have a look at the HTMLArea documentation.