HTML Documentation Panel Plugin 1.0 Component Specification

1. Design

The documentation for elements in the UML Tool is currently entered into a basic text box that has no formatting capabilities, as part of the Documentation Panel component. This component will provide the base for a new HTML-based documentation editor. This initial version will support rendering the HTML and basic editing, with bold, italics and underline supported for text. This component will be able to be configured in the UML Tool through configuration of the Documentation Panel.

This component provides a HTML editor HTMLDocumentationEditor which can be used by ModelElement class in the TCUML tool. The HTMLDocumentationEditor also is the main class of the component. It can handle the content with the type: XHTML. With the help of a toolbar, the editor can bold, italic, underline selected texts, and can choose font size and font name of the selected texts. It also can undo and redo the edits. It also provides keyboard shortcuts for bold, italic, underline, undo and redo actions. The toolbar is an instance of DefaultHTMLDocumentationEditorToolBar which inherits from HTMLDocumentationEditorToolBar.

1. Design Patterns

Strategy pattern –DefaultHTMLDocumentationEditorToolBar is an implementation of HTMLDocumentationEditorToolBar that can be plugged to HTMLDocumentationEditor.

2. Industry Standards

XHTML

1. Required Algorithms(updated)

- *1.3.1 Validate XHTML input*

This part depicts how to convert input to XHTML documents which lets the invalid tags to display as raw data. It refers to HTMLDocumentationEditor#validateInput(String input) method.

1. Get the factory for the W3C XML Schema language: factory = SchemaFactory.newInstance(“http://www.w3.org/2001/XMLSchema”);
2. Compile the schema by using the XHTML XSD like http://www.w3.org/2002/08/xhtml/xhtml1-strict.xsd , we assume it’s local path is xsdPath): Schema schema = factory.newSchema(new File(xsdPath));
3. Get the validate from the schema: Validator validator = schema.newValidator();
4. Create source object

```
//Get the validate from the schema
Validator validator = schema.newValidator();

//create an input source
```

```
InputSource inputSrc = new InputSource(new ByteArrayInputStream(input.getBytes()));

//create the SAX source
SAXSource saxSource = new SAXSource(inputSrc);
```

5. Try to validate the input: validator.validate(saxSource);
6. If not exception occurs, the input is a valid XHTML, return;
7. Else, if SAXException catches, set the content type to plain text: content.setContentType("text/plain");

- *1.3.2 Undo/Redo Management*

This section refers to how to implement undo/redo command in this component. This part is used for better understanding of Undo/Redo. In most cases, the developers do not need to care about this, since this is a regular way now.
To use undo/redo, Memento pattern is applied. The two roles of this pattern in this component is originator: the editor and a caretaker: the UndoManager. And the memento object to manage is the documentation of editor. UndoManager is the amazing
classes that keeps a Stack of UndoableEdits and lets you invoke them; by registering it as a Listener on the documen
of the editor, the Document will create the UndoableEdit objects and send them
to the UndoManager. Between them they do ALL the work!

Each time we edit the documentation like: bold the selected text, an UndoableEditEvent:
e is generated; we add it to the undoManager and then get the manager's undo/
redo names and set the update undo/redo action. Finally, we enable/
disable these buttons by asking the manager what we are allowed to do. This work is done through undoHandler. It catches the UndoableEditEvent:e, by invoking undoableEditHappened(e). It will do the following job:

1. call undoManager to add the last edit to the stack: undoManager.addEdit(e.getEdit());
2. Update the undo action command: undo.update(), it will do:

1. Determine whether the undo action can be applied: undoManager.canUndo();
2. If can undo the last edit, enable the undo action: setEnable(true);
3. Else, disable the undo action: setEnable(false);
3. Update the redo action command: redo.update(), it will do:

1. Determine whether the redo action can be applied: undoManager.canRedo();
2. If can redo the last edit, enable the redo action: setEnable(true);
3. Else , disable the redo action: setEnable(false);

If the user wants to undo the last edit, he can click on the undo button or by short keys. Then The undo catches the ActionEvent:e by calling actionPerformed(e), it will do the following:

1. undoManager will undo the edit by poping out the last edit from the stack: undoManager.undo();
2. Then update the undo action: update();

3. Update the redo action: redo.update();

If the user wants to redo the last edit, he can click on the redo button or by short keys. Then The redo catches the ActionEvent:e by calling actionPerformed(e), it will do the following:

1. undoManager will redo the edit by pushing back the last edit from the stack: undoManager.redo();
2. Then update the redo action: update();
3. Update the udo action: undo.update();

4. Component Class Overview

HTMLDocumentationEditor:

This is the main class of the component. It’s a HTML editor panel provided with this component. It has a JEditorPane instance as the text box, contained within a scrolling panel where the bars appear as needed from configuration. It is also possible to set the text box to wrap when the words reach the end of the line. Then, whenever focus is lost, the documentation changed is fired to all listeners. With the help of a toolbar, the editor can bold, italic, underline selected texts, and can choose font size and font name of the selected texts. It also can undo and redo the edits. It also provides keyboard shortcuts for bold, italic, underline, undo and redo actions.

This class is not thread safe – since the class is mutable, and the focus events are triggered by GUI actions which can occur at any time, for example possibly when the model target is being changed by a setTarget call, which may cause problems. It can also change the toolbar attached to the editor.

HTMLDocumentationEditorToolBar <<Abstract>>:

This abstract class lays the basic contracts of a tool bar for HTML documentation editor. It contains the following basic commands: cut, copy, paste, bold, italic, underline, undo and redo. It also contains keyboard shortcuts for the commands.

This class is not thread safe – since the class in mutable and it’s a SWING class.

UndoAction <<Inner>>:

This class inherited from AbstractAction is used to manage the undo action. It will undo the previous edit. It’s an inner class of HTMLDocumentationEditorToolBar.

This class is not thread safe – since the super class is not thread safe.

RedoAction <<Inner>>:

This class inherited from AbstractAction is used to manage the redo action. It will redo the previous cancelled edit. It’s an inner class of HTMLDocumentationEditorToolBar.

This class is not thread safe – since the super class is not thread safe.

UndoHandler<<Inner>>:

This class implementing the interface UndoableEditListener is used to manage the undoable edit. It will do the management by listening and handling the UndoableEditEvent through method #undoableEditHappened.It's an inner class of HTMLDocumentationEditorToolBar.

This class is thread safe – since it's immutable.

DefaultHTMLDocumentationEditorToolBar:

This is the default tool bar used by the editor. It inherits from the HTMLDocumentationEditorToolBar. Besides the basic commands from the super class, it also contains the commands to select font name, font size, and font color for texts.

This class is not thread safe – since the super class is not thread safe.

5. Component Exception Definitions

HTMLDocumentationEditorException:

Exception thrown when the classes cannot read the configuration properties, or the properties violate the conditions upon them in CS 3.2, or there're some errors thrown by other classes like Class.

This class is not thread safe – since the super class is not thread safe.

6. Thread Safety

Most of the classes in this component are not thread safe - the editor implementations are not protected from concurrent read/writes, as it contains non thread-safe members, and does not handle them in a thread safe manner. The only classes with thread safety are those which are read-only after construction: the UodoHandler classes. Actually, any SWING class is not thread safe. In order to use them in safe way, please refer to http://java.sun.com/products/jfc/tsc/articles/threads/threads1.html.

2. Environment Requirements

1. Environment

Development language: Java 1.5

Compile target: Java 1.5 / 1.6

1. TopCoder Software Components(updated)

- **Documentation Panel 1.0** – provides the base editor class: DocumentationEditor for inheritance.
- **Base Exception 2.0** – is used by custom exceptions defined in this component.
- **Configuration Persistence 1.0.1** - for loading configuration parameters.
- **UML Model Core 1.0** - for the ModelElement and related interfaces
- **Configuration API 1.0** – is used for reading the configuration of this component.
- **Uml model core extension mechanisms 1.0** – is used for set target model element

NOTE: The default location for TopCoder Software component jars is../lib/tcs/COMPONENT_NAME/COMPONENT_VERSION relative to the component installation. Setting the tcs_libdir property in topcoder_global.properties will overwrite this default location.

3. Third Party Components

None.

3. Installation and Configuration(updated)

1. Package Name

com.topcoder.gui.panels.documentation.plugins.html
com.topcoder.gui.panels.documentation.plugins.html. toolbars

2. Configuration Parameters

The following table describes the configuration for the constructor of HTMLDocumentationEditor (Please note: parameters with default values are **optional**, others are **required**).

Parameter	Description	Values
toolBarClass	The full class name of toolbar. The class must be a sub-class of HTMLDocumentationEditorToolBar.	String. Must be a valid class name. Default is: "com.topcoder.gui.panels.documentation .plugins.html.toolbars.Default HTMLDocumentationEditorToolBar".
XSDPath	The XHTML XSD file path used to validate the XHTML input	String. Must be a valid XHTML XSD file path.
vsbPolicy	an integer that specifies the vertical scrollbar policy	Int. Legal values are: ScrollPaneConstants. VERTICAL_SCROLLBAR_AS_NEEDED ScrollPaneConstants. VERTICAL_SCROLLBAR_NEVER ScrollPaneConstants. VERTICAL_SCROLLBAR_ALWAYS Default is ScrollPaneConstants. VERTICAL_SCROLLBAR_AS_NEEDED

hsbPolicy	an integer that specifies the horizontal scrollbar policy	Int. Legal values are: ScrollPaneConstants. HORIZONTAL_SCROLLBAR_AS_NEEDED ScrollPaneConstants. HORIZONTAL_SCROLLBAR_NEVER ScrollPaneConstants. HORIZONTAL_SCROLLBAR_ALWAYS Default is ScrollPaneConstants. HORIZONTAL_SCROLLBAR_AS_NEEDED
toolBarConfig	The configuration for toolbar.	ConfigurationObject. Cannot be null. Valid properties are listed as the following table.

The following table describes the configuration for the constructor of HTMLDocumentationEditorToolBar and DefaultHTMLDocumentationEditorToolBar(Please note: parameters with default values are **optional**, others are **required**).

Parameter	Description	Values
cutImage	The image icon for button of cut action command	String. Must be a valid image URL string. Cannot be null.
cutShortKey	The keyboard shortcuts for button of cut action command	Character. This shortcut will combine with CTRL (on Windows/Linux) or Command (on Macintosh).Default: X
cutTooltip	The tool tip for button of cut action command	String. Default is: “cut the text”.
copyImage	The image icon for button of copy action command	String. Must be a valid image URL string. Cannot be null.
copyShortKey	The keyboard shortcuts for button of copy action command	Character. This shortcut will combine with CTRL (on Windows/Linux) or Command (on Macintosh).Default: C

copyTooltip	The tool tip for button of copy action command	String. Default is: “copy the text”.
pasteImage	The image icon for button of paste action command	String. Must be a valid image URL string. Cannot be null.
pasteShortKey	The keyboard shortcuts for button of paste action command	Character. This shortcut will combine with CTRL (on Windows/Linux) or Command (on Macintosh).Default: V
pasteTooltip	The tool tip for button of paste action command	String. Default is: “paste the text”.
undoImage	The image icon for button of undo action command	String. Must be a valid image URL string. Cannot be null.
undoShortKey	The keyboard shortcuts for button of undo action command	Character. This shortcut will combine with CTRL (on Windows/Linux) or Command (on Macintosh).Default: Z
undoTooltip	The tool tip for button of undo action command	String. Default is: “undo the edit”.
redoImage	The image icon for button of redo action command	String. Must be a valid image URL string. Cannot be null.
redoShortKey	The keyboard shortcuts for button of redo action command	Character. This shortcut will combine with CTRL (on Windows/Linux) or Command (on Macintosh).Default: Y
redoTooltip	The tool tip for button of redo action command	String. Default is: “redo the edit”.
boldImage	The image icon for button of bold action command	String. Must be a valid image URL string. Cannot be null.
boldShortKey	The keyboard shortcuts for button of bold action command	Character. This shortcut will combine with CTRL (on Windows/Linux) or

		Command (on Macintosh).Default: B
boldTooltip	The tool tip for button of bold action command	String. Default is: “bold the text”.
italicImage	The image icon for button of italic action command	String. Must be a valid image URL string. Cannot be null.
italicShortKey	The keyboard shortcuts for button of italic action command	Character. This shortcut will combine with CTRL (on Windows/Linux) or Command (on Macintosh).Default: I
italicTooltip	The tool tip for button of italic action command	String. Default is: “italic the text”.
underlineImage	The image icon for button of underline action command	String. Must be a valid image URL string. Cannot be null.
underlineShortKey	The keyboard shortcuts for button of underline action command	Character. This shortcut will combine with CTRL (on Windows/Linux) or Command (on Macintosh).Default: U
underlineTooltip	The tool tip for button of underline action command	String. Default is: “underline the text”.
fontSizeConfig	The font size configuration.	Int[]. The elements of the array are: the start of the font size, the end of the font size, and the step from each size to the next one.

3. **Dependencies Configuration**

The configuration API must be set up correctly to allow the above configuration parameters to be read. Please consult its documentation for steps to use that component.

4. Usage Notes

1. Required steps to test the component

- Extract the component distribution.
- Follow Dependencies Configuration.
- Execute ‘ant test’ within the directory that the distribution was extracted to.

2. Required steps to use the component

Please see the demo.

1. Demo (updated)

1. *Usage of the API*

```
String namespace = "html_documentation_editor";

//First, prepare the configuration as described in Section 3.2
ConfigurationFileManager configManager = new ConfigurationFileManager();
ConfigurationObject config = configManager.getConfiguration(namespace);

//Then, create the editor. This will get a HTML editor with a tool bar.
HTMLDocumentationEditor editor = new HTMLDocumentationEditor(config);

//Or create the editor with namespace as parameter.
//namespace is used with ConfigManager
//HTMLDocumentationEditor editor1 = new HTMLDocumentationEditor(namespace);

//If the user wants to add some button corresponding to some action,
//like StyledEditorKit.AlignmentAction, he can do it like this:
//Add a the left align action button
editor.getToolBar()
.addToolBarButton(new ImageIcon("test_files/leftAlign.gif"),
        new StyledEditorKit.AlignmentAction("leftAlign", 0), 'L', "Left align the text");

//If the user wants to change the tool bar, he can do it like this
//Initialize another tool bar
//HTMLDocumentationEditorToolBar toolbar = TestHelper.genToolBar(namespace);

        //set the tool bar to the other tool bar inherited from HTMLDocumentationEditorToolBar
//editor.setToolBar(toolbar);
ModelElement modelElement = new AttributeImpl();

TaggedValue tv = new TaggedValueImpl();
TagDefinition tg = new TagDefinitionImpl();
tv.setType(tg);
modelElement.addTaggedValue(tv);
```

```
//Set input to the editor
editor.setTarget(modelElement);

//Get output from the editor
editor.getContentText();

//the output can also get from #focusLost
//by the classes which are interested in FocusEvent
FocusEvent event = new FocusEvent(editor, FocusEvent.FOCUS_GAINED);

editor.focusLost(event);

//Then, add the editor to a frame.
JFrame testFrame = new JFrame("Test");
testFrame.setPreferredSize(new Dimension(800, 600));
testFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

testFrame.add(editor);
testFrame.pack();
        testFrame.setVisible(true);
```

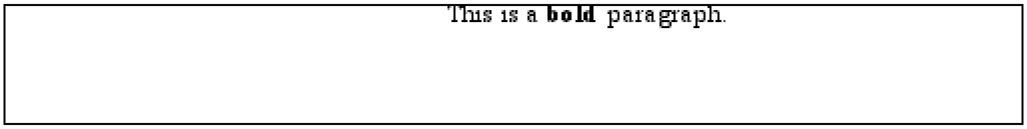
2. XHTML input and output

The following gives an example of valid XHTML input and its output in the editor.

The input:

```
<html>
<head>
</head>
<body>
<p>This is a <b>bold</b> paragraph. </p>
</body>
</html>
```

The output:



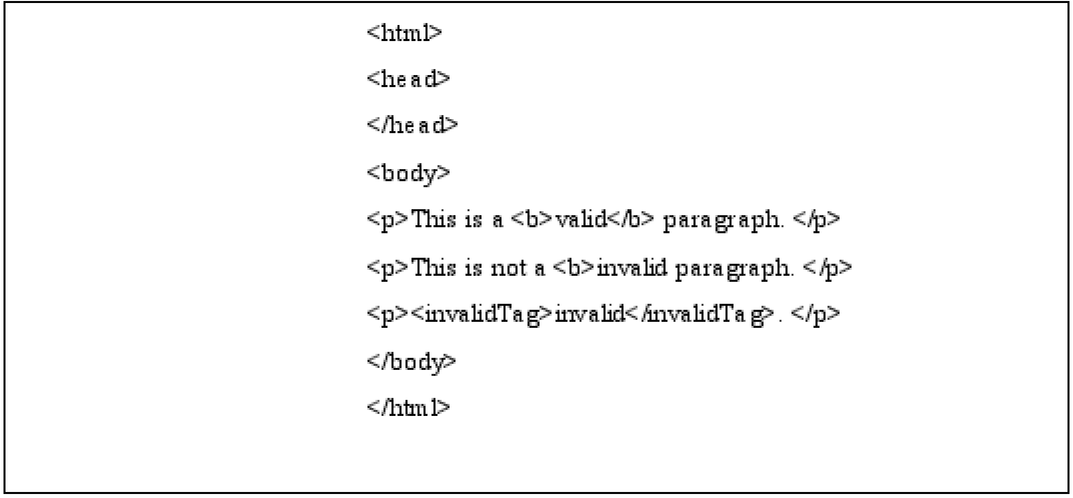
This is a **bold** paragraph.

Since it's a valid XHTML input, so display it as web page.

The following gives an example of invalid XHTML input and its output in the editor.
The input:

```
<html>
<head>
</head>
<body>
<p>This is a <b>valid</b> paragraph. </p>
<p>This is not a <b>invalid paragraph. </p>
<p><invalidTag>invalid</invalidTag>. </p>
</body>
</html>
```

The output:



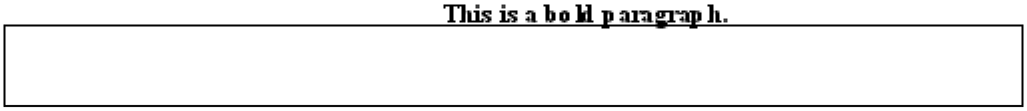
```
<html>
<head>
</head>
<body>
<p>This is a <b>valid</b> paragraph. </p>
<p>This is not a <b>invalid paragraph. </p>
<p><invalidTag>invalid</invalidTag> . </p>
</body>
</html>
```

From the output, the second does not, since the tag ‘<b>’ does not have a close tag. While the third paragraph does not have a tag supported by XHTL, so display it as raw data.

From the output, the second does not, since the tag ‘<b>’ does not have a close tag. While the third paragraph does not have a tag supported by XHTL, so display it as raw data.

3. Interact with User

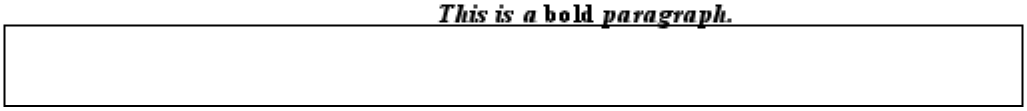
   This part demonstrates the interaction with users. Here, we show one typical scene : the user first bold the text(valid text displayed from the 4.3.2), then italic it, then click undo button for one time, then underline the text, finally click the redo button. The following gives the changes of the text:



This is a bold paragraph.

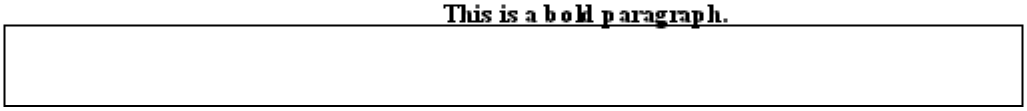
Bold the text:

Italic the text:



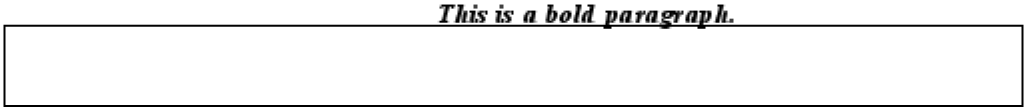
This is a bold paragraph.

Click the undo button:



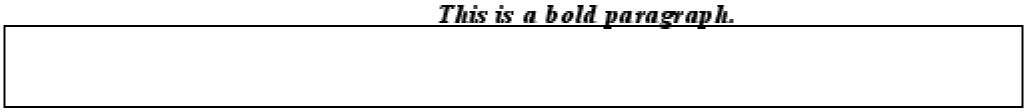
This is a bold paragraph.

Click the redo button:



This is a bold paragraph.

Underline the text:



This is a bold paragraph.

4. *Plug the editor to documentation panel*

   One way is just create the documentation panel with name space specified.

   ```
   DocumentationPanel panle = new
   DocumentationPanel(namespace);
   ```

   In this way, the constructor of DocumentationPanel will use reflection to create the editor through the constructor: HTMLDocumentationEditor(nameSpace).

   Another way is to use the editor as an argument of the DocumentationPanel.

   ```
   DocumentationPanel panle = new DocumentationPanel(editor,
   tageEditor);
   ```

5. **Future Enhancements**

Links, tables and lists will all be added to the component in the future.