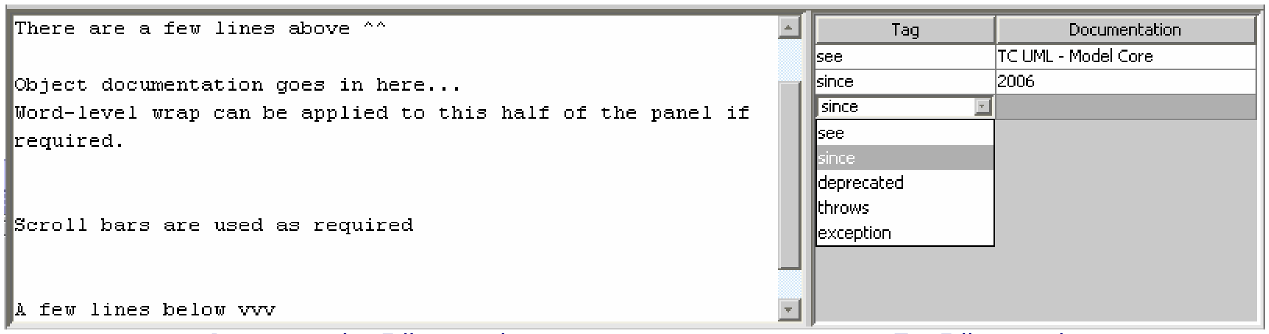
Documentation Panel 1.0 Component Specification

1. Design



```
There are a few lines above ^^

Object documentation goes in here...
Word-level wrap can be applied to this half of the panel if
required.


Scroll bars are used as required


A few lines below vvv
```

Tag	Documentation
see	TC UML - Model Core
since	2006
since	

Documentation Editor section *Tag Editor section*

The Documentation Panel component is a Swing-based GUI panel, which allows users to manage documentation and 'tags' attached to elements within a diagram. Any ModelElement object can act as the target for documentation, and this panel enables users to generate documentation/tag events detailing when and how the element's documentation is to be modified. The panel itself is merely a container for two plug-in editors, for documentation and tags.

On the left half of the panel lies a Documentation editor plug-in - this panel is used to edit the general documentation found on the target element. Provided with this component is a simple default editor comprising of text box, word wrapped if desired, which sends update signals whenever focus leaves the box.

The right side of the panel is the more complex Tag editor plug-in, handing all documentation tags that have been attached to the element (as well as parameter documentation for methods). This allows users to add, edit, and remove [name => documentation] pairs associated to the element. A default tag editor is provided with this component, which allows the tag names to be selected from a drop-down combo box of options, tag documentation to be updated by entering text in the corresponding row of a table, and deleted by removing associated text. Events are fired whenever a tag is added, changed, or removed.

In addition to these two, the Panel itself gives the users the ability to register listeners which are called whenever the modification events are fired, by implementing provided interfaces. Also, methods are provided to make it possible for an application to hide either the tag plug-in, or to blank out the whole panel, in times when documentation should not be accessible.

1.1 Design Patterns

Delegate: Apart from basic GUI layout, most processing (e.g. listeners and display) attached to a DocumentationPanel is delegated to its two xxEditor members.

Command: Each xxEvent class wraps up information about a documentation event before passing it to listeners to process how they deem is necessary.

Observer: The editors maintain lists of xxListener observer classes, notified whenever documentation is changed.

MVC: The Model-View-Controller is used within the Swing in this component - for example, the tag table has an underlying model class, which is then displayed by a JTable.

1.2 Industry Standards

Swing components for Java GUI.

1.3 Required Algorithms

The only complex ‘algorithm’ is that for construction/configuration:

```
Construct DocumentationPanel:
    Read in Object factory namespace, creating a factory instance using a config
    manager specification factory.

     Read Object factory key for DocumentationEditor
     Obtain a DocumentationEditor from the Object factory, by default using the
    String-only constructor:
           Read line-wrap option
           Set up content text area, place in scroll panel, place in self.

     Read Object factory key for TagEditor
     Obtain a TagEditor from the Object Factory, by default using the
     String-only constructor:
           Read optional column names / prompts parameters
           Set up the table model, using optional parameters
           Set up the table using the model, place in scroll panel, place in self.

           Read in namespace for TagOptionManager
           Construct the TagOptionManager:
                 For all identifier keys:
                       Read in options, add to map
                       Read in links, add to map
```

1.4 Component Class Overview

DocumentationPanel:

A DocumentationPanel is a SWING JPanel, allowing a user two ways to edit the documentation attached to a Model Element within a UML Model: by editing the main documentation attached directly to the element, or by editing the "tags" (name=>value pairs) that have been attached. In addition, operation elements can have their parameter documentations edited along with their tags. The panel does this by initializing a JSplitPane, then displaying a DocumentationEditor plugin on the left and a TagEditor plug-in on the right. The plug-ins are set on construction using configuration data, and interaction is possible by adding listeners which have methods called whenever documentation is created/updated/deleted. The DocumentationPanel also provides a way to change the current target whose documentation is to be managed, as well as two ways to hide sections of the editor when they are not needed.

DocumentationEditor << abstract >>:

Abstract superclass that defines all methods required for a documentation editor to be used within a DocumentationPanel. A DocumentationEditor is a JPanel which gives the user the ability to modify the documentation attached to a ModelElement within a UML model. This abstract class stores the target element, and also manages all listener classes that are waiting for notifications of when documentation is edited.
Note that this, and all extending subclasses, should not actually modify the documentation of their target in any way, only firing events when the user wants changes. They should also all have single - String constructors, to allow simple construction via reflection by a DocumentationPanel

DefaultDocumentationEditor:

A simple, default tag editor panel provided with this component - It is simply a large 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.

TagEditor << abstract >>:

Abstract superclass that defines all methods required for an editor of tags that is to be used within a DocumentationPanel. A TagEditor is a JPanel which gives the user the ability to modify 'tags' (name=>documentation pairs) that are attached to a ModelElement within a UML model. This abstract class stores the target element, and also manages all listener classes that are waiting for notifications of when element tags are edited.
Note that this, and all extending subclasses, should not actually modify the documentation of their target element's tags in any way, only firing events when the user wants changes. They should also all have single - String constructors, to allow simple construction via reflection by a DocumentationPanel

DefaultTagEditor:

A simple, default tag editor panel provided with this component - It contains a two-column table, where tags can be edited from a drown-down combo-box on the left, then have their documentation modified on the right. This is obtained by controlling a custom table model, as well as an object to manage all the tag options available.

DefaultTagTableModel:

Custom table model class to represent the information stored in a tag table. The model is a two dimensional, Nx2 matrix of Strings - the first column compromising the names of tags, and the second their corresponding documentation. The last row of the table is special, as only the tag name is creatable, all other rows only have their 2nd column (documentation) editable. This default tag table model contains some customizable values, to allow the prompts and column names to be changed based on configuration.

TagOptionManager:

A TagOptionManager is responsible for handling the customization of tag options for each registered class type. Each different element within the model may have its own set of appropriate tags (e.g. "author", "see", "throws", ...) - these lists are customizable through configuration, and it is up to this manager to organize parsing the configuration into option lists for each identifiers given.

DocumentationListener << interface >>:

Interface defining the three methods that a documentation listener should listen for - Corresponding to documentation creation / update / deletion, each takes a single DocumentationEvent parameter describing the properties of the event that was fired. Implementations are to handle their own thread safety if it is required.

DocumentationEvent:

A DocumentationEvent contains all information when an element's documentation gets modified. It has a required attribute, the model element whose documentation is being changed. Along with this is stored the previous and current documentation, possibly null if the documentation was created/deleted.

TagListener << interface >>:

Interface defining the three methods that a tag listener should listen for - Corresponding to tag creation / update / deletion, each takes a single TagEvent parameter describing the properties of the event that was fired.

TagEvent:

A TagEvent contains all information when a tag's documentation gets modified. The two required attributes are the model element whose tag is being changed, as well as the identification name of the tag. Along with this is stored the previous and current documentation of the tag, possibly null if the tag was created/deleted.

1.5 Component Exception Definitions

DocumentationPanelConfigurationException:

Exception thrown when any documentation cannot load required information from configuration. This can either be if required documentation is missing or invalid, or if the configuration itself cannot be read.

UnknownElementTypeException:

Exception thrown when an application tries to use an element in a panel, and the element's type is unrecognized by that panel.

1.6 Thread Safety

Most of the classes in this component are not thread safe - the editor implementations are not protected from concurrent read/writes, and so the overall Documentation pane is also not thread safe, 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 Event classes as well as the Tag option manager.

2. Environment Requirements

2.1 Environment

- Java 1.5 for compilation and development

2.2 TopCoder Software Components

- UML Model Core 1.0, for the ModelElement and related interfaces
- Base Exception 1.0, for the BaseException class
- Configuration Manager 2.1.5, for loading configuration parameters.
- Object Factory 2.0.1 is required for creation of the panel's editors.

3. Installation and Configuration

3.1 Package Name

com.topcoder.gui.panels.documentation
com.topcoder.gui.panels.documentation.event
com.topcoder.gui.panels.documentation.plugins

3.2 Configuration Parameters (required unless otherwise specified)

Parameter	Description	Values

objectFactoryNamespace	Namespace to use for Object Factory	Valid Object Factory path.
documentEditorKey	Key used to obtain the document editor from the object factory.	Valid Object Factory key.
tagEditorKey	Key used to obtain the tag editor from the object factory.	Valid Object Factory key.
Editor configuration parameters:		
documentEditorNamespace/ wordwrap	Whether word wrapping is to be applied to the documentation editor. [Optional, defaults to off.]	"yes" if word wrap is desired.
tagEditorNamespace/ tagprompt	Prompt to display on the combo box of available tag names. [Optional, defaults to "Insert new tag…"]	Any string.
tagEditorNamespace/ valueprompt	Prompt to use for initial documentation of new tags added. [Optional, defaults to "new value"]	Any string
tagEditorNamespace/ columnnames	The two names to use as column headers in the tag table. [Optional,defaults"Tag","Documentation"]	Two strings.
tagEditorNamespace/ tagoptionNamespace	Namespace used to construct a tag option manager instance.	Valid namespace string
tagoptionNamespace/ **identifierkey/**options	List of options to attach to the identifier given by **identifierkey** [Optional, defaults to empty array]	0..* Strings
tagoptionNamespace/ **identifierkey/**links	List of identifier links to attach to the identifier given by **identifierkey** [Optional, defaults to empty array]	0..* Strings

See sample_config.xml for an example configuration file.

3.3 Dependencies Configuration

1. The configuration manager, base exception and Object factory must be set up correctly to allow the above configuration parameters to be read. Please consult its documentation for steps to use that component.

2. uml_models.jar in /test_files

3. jfcunit.jar from http://jfcunit.sourceforge.net

4. jakarta-regexp-1.4.jar from http://jakarta.apache.org/site/downloads/downloads_regexp.cgi

4. Usage Notes

4.1 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.

4.2 Required steps to use the component

Covered in the demo code provided below.

4.3 Demo

Simple execute \test_files\demo\demo.bat to launch the demo. It shows what the GUI panel may look like.

Demonstration code has been provided for examples of how the component can be used:

\src\java\tests\driver\PanelDriver.java

\src\java\tests\com\topcoder\gui\panels\documentation\event\DemoTagListener.java

\src\java\tests\com\topcoder\gui\panels\documentation\event\DemoDocumentationListener.java

Included is the ability to show/hide the whole panel, or tag panel, and events are printed to standard out with descriptions of what has happened.

The main features are shown below:

```
// Create the panel directly:
docPanel =
   new DocumentationPanel(“docPanelNamespace”); // sample namespace

// Manipulate its visibility:
docPanel.showWholeEditor(false); // hide whole panel
docPanel.showTagEditor(false);   // optionally hide only tags

// Set the element being documented:
try{
   OperationImpl myTarget = ...;   // selected by user
   docPanel.setTarget( myTarget );
} catch(Exception e){
   // no exceptions thrown, instead the editors are hidden
}

// show the panel again: (no side effects if already visible)
docPanel.showWholeEditor(true);
docPanel.showTagEditor(true);
// e.g. if the target is a OperationImpl, the tag options will now appear
//   as “see”, “since”, “deprecated”, “throws” and “exception”

// Attach some listeners:
// some DocumentationListener implementation
docPanel.addDocumentationListener(new DemoDocumentationListener());
// some TagListener implementation
docPanel.addTagListener(new DemoTagListener());
```

```
// =============
// Listener code
public void tagCreated(TagEvent event) {

      JOptionPane.showMessageDialog(null, event.getTagName() + " : "
+ event.getCurrentDocumentation(), "Tag event : created",
JOptionPane.INFORMATION_MESSAGE);

   }
```

5. Future Enhancements

The most obvious enhancements would be to provide other editor panels that can be plugged into the DocumentationPanel. These may include spell-checking panels, pop-out panels, or a tag panel that does not use a 2-column table.

Eventually, it may be worthwhile to incorporate the use of the Generic Event Manager component to handle the tag and documentation event management. Currently, the event triggers are specialized enough that the reflection overhead is not warranted, but this could be useful if combined within a larger application already using the Generic Event Manager.