Diagram Viewer 1.0 Component Specification

1. Design

The Diagram Viewer component provides a SWING tabbed panel that will display the diagrams. The component will also provide the general diagram panel (that can be zoomed and can have a background grid) and the input text control used in GUI applications to enter text for different elements present in the diagram.

This design primarily aims at high flexibility and customization, it can be seen in the following points:

1. The DiagramViewer has several configurable properties, such as: max number of open tabs, diagram background color, etc. These properties can be initialized from configuration manager in constructor, and also be accessed by getter and setter, the application can customize them after startup at any time. All the properties are optional and have their own default value.

2. The close tab button and zoom spin button are also configurable, the button name and icon can be configured through configuration manager. And since only actions are defined for the functionality, we can provide other customized buttons in future without worrying about the logic.

3. The background grid is rendered by a configurable renderer, an interface is defined for it, so we can provider different implementations in future.

4. The event sub package is designed to separate event process in different handlers, this make the process logic clear and easy to understand.

5. A bottom bar was provided with DiagramViewer, the zoom spin buttons are added on it, the application can also add some other widgets on it to provide more functionality, methods are provided to show or hide it.

1.1 Design Patterns

Observer Pattern - System events are listened in this component, and also custom events are triggered for application to listen.

Template Method Pattern - AddNewElementHandler and SelectElementsHandler provide different implementations for mouse event process methods from MouseDragHandler class.

Strategy Pattern - The DiagramView can show background grid, this is done by using GridRenererer interface. Different implementations can provide different rendering strategy.

Factory Method Pattern - The DiagramViewer acts as a way of factory for creating DiagramView.

MVC Pattern - The DiagramViewer represents Model, the DiagramView represents View, and the event handlers represents Controller.

1.2 Industry Standards

SWING/JFC

1.3 Required Algorithms

Note: The handling of exceptions has not been shown below, the developer has to take care of handling the exceptions during development. Details about how to handle exception can be get in the Poseidon doc.

1.3.1 Initialization of DiagramViewer (constructor of DiagramView):

The DiagramViewer is a complex class that has several configurable properties, some of them are loaded from ConfigManager and some of them are created by using ObjectFactory. After initializing properties, the constructor should create some event handlers (in event sub package) and register them. Since the DiagramViewer is a concrete JComponent, the constructor also adds buttons and panels to the viewer.

The following steps show what and how to do:

I. Loading properties from ConfigManager:

Some properties can be loaded directly from ConfigManager, for these properties, initializing them in this way:

1. Get an instance of ConfigManager.

2. Load the configuration information for the specified namespace.

3. Get the string value of the property using method ConfigManager#getString.

4. If the property is boolean or integer, using Boolean.parseBoolean or Integer.parseInt to parse the string got in last step

For example, the property field DiagramViewer.maxOpenTabs can be initialized like this:

```
ConfigManager cm = ConfigManager.getInstance();

String value = cm.getString(namespace, "max_open_tabs");

maxOpenTabs = Integer.parseInt(value);
```

II. Instantiating field instances using ObjectFactory:

The fields need to be created by ObjectFactory are diagramViewCache and gridRenderer:

1. Load object factory namespace used by this component from the ConfigManager.

2. Initialize the ObjectFactory instance for that namespace.

3. Read the property name for the field from ConfigManager.

4. Create object using ObjectFactory#createObject method.

For example the diagramViewCache can be created in this way:

```
String factoryNamespace = ConfigMan-
ager.getInstance().getString(namespace, "object_factory_namespace";

ObjectFactory of = new ObjectFactory(new ConfigManagerSpecifica-
tionFactory(factoryNamespace));

String cacheName = ConfigManager.getInstance().getString(namespace,
"diagram_view_cache");

diagramViewCache = of.createObject(cacheName);
```

III. Creating functional widgets (close tab and zoom buttons, and zoom text field):

There are three functional buttons in this component: close tab button on the upper right corner, zoom in/out buttons on the lower right corner, they are created with the Action implementations of this component:

1. Read the action namespace from the ConfigManager.

2. Create an Action instance (CloseTabAction or ZoomAction).

3. Create a JButton with the Action instance.

For example the close tab button can be build in this way:

```
String actionNamespace = ConfigMan-
ager.getInstance().getString(namespace,
“close_tab_action_namespace”;

CloseTabAction action = new CloseTabAction(actionNamespace);

JButton closeTabBtn = new JButton(action);
```

And there is also a text field that is used to change zoom factor for current active tab, the class ZoomTextFieldAction is provided as a event handler for this text field, so just create the text field and register ActionListener for it:

```
JTextField textField = new JTextField();

ZoomTextFieldAction zoomTextFieldAction = new ZoomTextFieldAc-
tion(this, textField);

textField.addActionListener(zoomTextFieldAction);
```

IV. Building the component panel:

After initializing the inner fields by using ConfigManager and ObjectFactory, it is the time to build the GUI panel, the DiagramViewer provide a tabbed pane to display multiple diagrams inside tabs, and need to show a close tab button on the upper right corner, zoom in/out buttons on the lower right corner. Unforunately, the JTabbedPane can not be customized much in java 1.5, so we can not just let the DiagramViewer inherit the JTabbedPane.

To obtain the required feature, this design uses a BorderLayout to divide the viewer’s panel into three parts: top, center and bottom. The close tab button is added to the top part at the rightmost place, the JTabbedPane is added to the center and fill it, the zoom in/out buttons is added to a bottom bar, which is JPanel, and itself is added to the bottom part.

Build the panel in this way:

1. Set the layout manager of DiagramViewer to BorderLayout.

2. Create a JPanel to hold the close tab button, and add the button to the rightmost corner of it.

3. Add the panel to BorderLayout.NORTH.

4. Initialize the bottomBar to a new JPanel, and add the zoom in/out buttons and zoom text field to the rightmost corner of it.

5. Add the bottomBar to BorderLayout.SOUTH.

6. Add the tabbedPane to BorderLayout.CENTER.

1.3.2 Render DiagramView:

There are two points needed to be noted when rendering DiagramView: background grid and drag rectangle.

I. Background grid

Whether to show background grid depends on the field backgroundGridVisible of DiagramViewer, if it is set to true, the DiagramView should show grid using an implementation of GridRenderer interface.

The grid in the active area should be painted differently (lighter) than the grid outside the active area, active area corresponds to the actual Diagram, see Poseidon for reference.

A default renderer is provided in this design named SimpleGridRenderer, it renders the grid in this way:

1. Record the original color.

2. Compute the number of rows and columns of grid according to size of grid and DiagramView.

3. Set color to gray and draw grid as the whole area is inactive.

4. Compute the active area according to the Diagram corresponds to the DiagramView.

5. Set color to black and draw grid in active area.

6. Restore the color to original one.

II. Drag Rectangle

The DiagramView will draw a rectangle using a dashed line if the mouse is dragged on it. This option is for visually selecting elements or adding new element.

There is an inner Rectangle instance in the DiagramView, it will be updated when mouse is being dragging, and an inner flag dragRectangleVisible to decide whether the rectangle is shown.

React to the mouse event in this way (MouseDragHandler):

1. When mouse is pressed on the DiagramView, set dragRectangleVisible to true, record the mouse position

2. When mouse is dragged:

2.1 Compute the rectangle using the new mouse position and the old position:

New Position: (x0, y0), Old Position: (x1, y1)

```
draggingRectangle.setBounds(Math.min(x0, x1), Math.min(y0, y1),
Math.abs(x1 - x0), Math.abs(y1 - y0));
```

2.2 Update the inner rectangle of DiagramView, since the flag has already been set to true, the DiagramView draws the rectangle using a dashed line.

3. When mouse is released, set the flag dragRectangleVisible to false.

1.3.3 Trigger zoom event:

The diagram view in this component can be zoomed by using ZoomPanel, each DiagramView is contained as a child of ZoomPanel, the ZoomPanel will do all the job needed for zooming.

Note the zoom action is not done in this component, it only provided methods to register listeners that have interest in zoom event, the event will be triggered in this component, it is the responsibility of the listener to do the final zoom.

To trigger a ZoomEvent, using method DiagramViewer.fireZoom with one argument as the increment of the zoom factor.

There are two ways to trigger a ZoomEvent:

I. Using zoom in/out button (ZoomAction)

When the button is clicked, it triggers a ZoomEvent. The increment of the zoom factor is specified by a configurable property (ZoomAction.increment)

II. Using mouse wheel while pressing Ctrl (ZoomHandler)

This handler is an implementation of MouseWheelListener, so it is invoked when mouse wheel is rotated. It is automatically registered to DiagramView in the constructor.

When the mouse wheel is rotated while key “Ctrl” is being pressed, it triggers a ZoomEvent, the increment can be configured using property “zoom_rotate_increment”.

1.3.4 Trigger scroll event:

When the DiagramView is scrolled (in fact it is ZoomPanel that is scrolled), a scroll event is triggered.

This is done by registering a AdjustmentListener (ScrollTrigger) to the scroll bars (both vertical and horizontal) of the ZoomPanel. It is automatically done when constructing ZoomPanel in method DiagramViewer#openDiagram.

1.3.5 Select elements in DiagramView:

The DiagramViewer maintains a list of selected elements inside the diagram, and methods are provided to add or remove elements from the list.

One way to select elements is using mouse to click the element to be selected, then the application should tell the DiagramViewer to select the element by calling addSelectedElement method.

Another way to select is dragging a selection rectangle on the DiagramView, all the elements intersect the rectangle will be selected.

This component draws the selection rectangle in DiagramView, and triggers selection rectangle changed event in SelectElementsHandler (registered to DiagramView automatically in constructor as a mouse listener). Those who have interest in such event can implement a SelectionListener and register it to DiagramViewer.

A default listener is provided to check which elements intersect the selection rectangle, and update the list of selected elements in DiagramViewer.

The checking is done in this way:

```
DiagramView view = (DiagramView) event.getSource();

Rectangle rectangle = event.getSelectionRectangle()
```

```
for each element in DiagramView :
//using method getComponentAt(i)
      if (rectangle.intersects(element.getBounds))
            view.getViewer().addSelectedElement(element);
      else
            view.getViewer().removeSelectedElement(element);
```

The developers are encouraged to choose more efficient method.

1.3.6 Add new element to DiagramView and trigger AddNewElementEvent:

There are three ways to add a new element to DiagramView, first is by clicking mouse on it, second is by dragging a bounds rectangle to specify bounds for new element, and third is by dragging and dropping an element to the DiagramView.

The dragging and dropping way is done by set a transfer handler to the DiagramView, the handler should transform the actual element from Transferable and added it to the DiagramView, so the DiagramView overrides the setTransferHandler method to prohibit null value.

To use which way is according to DiagramViewer's state, if the state is DiagramState.ADD_ELEMENT_BY_CLICK the way is adding by clicking, if the state is DiagramState.ADD_ELEMENT_BY_DRAGGING_RECTANGLE, the way is adding by dragging.

Before changing the state to one of the two above, the type of the new element must be specified by calling method DiagramViewer#setNewElementType. The DiagramViewer records the type in inner field newElementType.

Trigger an AddNewElementEvent after either clicking or dragging, then the listeners who has interest with such event should add the new element eventually.

Each time after the AddNewElementEvent is triggered, set the DiagramViewer#newElementType to null.

1.3.7 Creating Diagram Views for specified Diagram (DiagramViewer#createDiagramView):

The DiagramViewer has a configurable field diagramViewCache, this is an implementation of Cache (Simple Cache 2.0), this cache is used when creating DiagramView, the creating process acts in this way:

```
DiagramView view = diagramViewCache.get(diagram);
if (view != null) return view;
view = new DiagramView(diagram, this);
cache.put(diagram, view);
return view;
```

1.4 Component Class Overview

DiagramViewer:

This a concrete JComponent that can display multiple DiagramViews inside tabs, it uses an inner JTabbedPane to provide the tab functionality.

It has server configurable properties, such as max number of open tabs, whether to show background grid, whether to display tab title in full version, etc. See fields doc for details.

It can create instances of DiagramView for specified Diagram objects, and has a cache object to cache the DiagramView instances, the cache functionality is obtained by using Simple Cache 2.0 component, so the cache is configurable through configuration manager.

It maintains a list of selected elements, and has methods to update the list, also one method is provided to test if a element is selected.

It has a bottom bar that can be added some widgets, the bar itself is an instance of JPanel, and can be accessed by getter, application can add other widgets on it if needed.

Several custom event listener is automatically registered to it (or to the ZoomPanel created by it), see event sub package for detail.

This class is mutable and not thread safe.

DiagramView:

This is a concrete JComponent that will correspond to Diagram from Diagram Interchange component, The diagram will have a null layout manager, so the elements will be added according to the (x,y) coordinates.

It automatically register AddNewElementHandler, SelectElementsHandler, PopupMenuTrigger in ctor to provide functionalities such as adding new element by mouse click/drag, showing popup menu when mouse is right clicked.

It can not be constructed directly using new statement outside this package, to create a new instance using method DiagramViewer#createDiagramView.

This class is mutable so not thread safe.

DiagramState (enumeration):

This is an enumeration that stands for states of DiagramViewer.

The DiagramViewer has three states currently:

1) ADD_ELEMENT_BY_CLICK: user can add new element to the DiagramView by clicking on it.

2) ADD_ELEMENT_BY_DRAGGING_RECTANGLE: user can add new element to the DiagramView by dragging a bounds rectangle on the it.

3) SELECT_ELEMENT: user can select elements on the DiagramView by dragging a selection rectangle on it.

This is enumeration so is thread safe.

CloseTabAction:

This is an action class to be used to create a button on the bottom bar of the DiagramViewer, this button is used to close the currently enabled diagram tab..

This class is immutable, but the super class is mutable so this class is not thread safe.

ZoomAction:

This is an action class to be used to zoom in/out a ZoomPanel, it has a field increment that indicates how to compute the new zoom factor when zooming, if this field has a posi-

tive value, it means this is a zoom in action, else if the field has a negative value, it means this is a zoom out action.

This class is immutable, but the super class is mutable so this class is not thread safe.

TextInputBox:

This is a text input tool that is displayed as a popup, it appears as a text area that can accept user's input, and reacts the "enter" key to accept the text inputed into it and triggers an event.

It will be used in the DiagramView to receive user's input content.

It extends the JPopupMenu to obtain the popup feature, and it contains a JTextArea to get the input from the user.

It can be used as a single line text field, or automatically generate new line when "Ctrl + Enter" typed, this feature depends on the value of field newLineAccepted.

This class is mutable so not thread safe.

GridRenderer (interface):

This interface defines the contract that every background renderer must follow.

It contains only one method to render background grid for the DiagramView.

This class changes the state of the DiagramViewer so is not thread safe.

SimpleGridRenderer:

The default implementation of SelectionListener that checks which elements intersect the rectangle and updates the selected elements list.

This class changes the state of the DiagramViewer so is not thread safe.

subpackage: com.topdoer.gui.diagramviewer.event

ScrollListener (interface):

The listener interface for receiving scroll events occurred when user scrolling a DiagramView.

The class that is interested in processing a sroll event implements this interface, and the object created with that class is registered with a DiagramViewer instance, using the method addSelectionListener. When the scroll event occurs, that object's diagramViewScrolled method is invoked.

The implementations of this interface are not required to be thread-safe.

They could be used in a thread safe manner in this component.

ScrollEvent:

This event indicates that a DiagramView is scrolled, the DiagramView can be retrieved by getSource().

This class changes the state of the DiagramViewer so is not thread safe.

AddNewElementListener (interface):

The listener interface for adding new element event occurs in the DiagramView.

The class that is interested in processing the event implements this interface. The object created with that class is registered with a DiagramViewer. When the the DiagramViewer has prepared to add new element that object's addNewElement method is invoked.

The new element is added to the DiagramView by such a listener finally.

The implementations of this interface are not required to be thread-safe.

They could be used in a thread safe manner in this component.

AddNewElementEvent:

This event indicates that a new element should be added to the DiagramView.

It is triggered when the state of the Diagram Viewer is Diagram-State.ADD_ELEMENT_BY_CLICK or Diagram-State.ADD_ELEMENT_BY_DRAGGING_RECTANGLE, and the user finished the adding action (either by clicking or by dragging a rectangle on the DiagramView, the listener who is interest in this event should add the element to the DiagramView finally.

It has tow properties, one represents the type of the new element, another represents the DiagramView. Note, the DiagramView can be retrieved by method getSource().

This class is immutable and thread safe.

TextInputListener(interface):

The listener interface for receiving text input event occurs in the TextInputBox.

The class that is interested in processing the event implements this interface. The object created with that class is registered with a TextInputBox.

When the text is entered (by pressing "Enter") that object's textEntered method is invoked.

When the text input is cancelled (by pressing "Esc") that object's textCancelled method is invoked.

The implementations of this interface are not required to be thread-safe.

They could be used in a thread safe manner in this component.

TextInputEvent:

An event which indicates that a TextInputBox has finished or cancelled its text input process.

It contains only one property which is the text inputed in the TextInputBox.

This class is immutable so is thread safe.

ZoomListener (interface):

The listener interface for receiving zoom events after a DiagramView is zoomed.

The class that is interested in processing a zoom event implements this interface, The object created with that class is registered with a DiagramViewer instance, using the method addSelectionListener.

After the one of the DiagramView hold by that DiagramViewer is zoomed, that object's diagramViewZoomed method is invoked.

The implementations of this interface are not required to be thread-safe.

They could be used in a thread safe manner in this component.

ZoomEvent:

This event indicates that a DiagramView is zoomed in (or out).

It is triggered when the user clicks the zoom spin buttons on lower right corner of DiagramViewer.

It has only one property represents the new zoom factor.

Note, the ZoomPanel can be retrieved by method getSource(), the DiagramView can be retrieved as child of the ZoomPanel, and the Diagram can be retrieved by method DiagramView.getDiagram().

This class is immutable and thread safe.

SelectionListener(interface):

The listener interface for receiving selection events occurs in the DiagramView.

The class that is interested in processing a selection event implements this interface, and the object created with that class is registered with a DiagramViewer (which holds the instances of DiagramView), using the method addSelectionListener.

When the selection event occurs, that object's selectionRectangleChanged method is invoked.

The implementations of this interface are not required to be thread-safe.

They could be used in a thread safe manner in this component.

SelectionEvent:

This event indicates that the selection rectangle of a DiagramView is changed.

It is triggered when the user drags mouse on the DiagramView when the state of DiagramViewer is DiagramState.SELECT_ELEMENT.

It has only one property that is a rectangle which represents the selection rectangle.

This class is mutable (the Rectangle is mutable) so is not thread safe.

SelectionHandler:

The default implementation of SelectionListener that checks which elements intersect the rectangle and updates the selected elements list.

This class is immutable and not thread safe.

TextInputBoxKeyHandler:

This is an event listener which listens to key event for the TextInputBox.

When the user is typing in the TextInputBox, this listener will resize the box when needed (if the text in the box is longer than the box's width), and trigger TextInputEvent when key "Enter" or "Esc" is typed.

This event listener will be registered to the TextInputBox in this component automatically.

This class is immutable, and thread-safe.

MouseDragHandler:

This is an event listener which monitors mouse dragged action.

It maintains a drag-rectangle when mouse is being dragged, this rectangle can be used to draw a selection rectangle when selecting or a bounds rectangle when adding new element to the Diagram View.

This is the base class of other two listener, one is for adding new element and another is for selecting element.

This class is immutable, but it has a Rectangle field which are mutable, and the state of the Rectangle field is modified in several methods, so this class is not thread safe.

AddNewElementHandler:

This is an event listener which monitors mouse event for adding new element to the Diagram View.

There are two ways of adding elements:

1) When the mouse is clicked, this is processed in the method mouseClicked.

2) After the mouse draws a bounds rectangle for the element, this is processed in method mouseReleased.

To choose which way depends on the state of the Diagram Viewer (Diagram-Viewer#getState()).

This class is immutable, but its super class is not thread safe so is it.

SelectElementsHandler:

This is an event listener which monitors mouse dragged action for selecting elements.

It inherits MouseDragHandler to record drag rectangle when dragging mouse. When mouse dragged event occurs, it checks which elements intersect the rectangle and updates the list of 'selected' elements of the DiagramViewer.

This class is immutable, but its super class is not thread safe so is it.

ScrollTrigger:

This is an event listener which listens to adjustment event for a JScrollBar.

If the event occurs, it means the ZoomPanel is being scrolled, a ScrollEvent will be triggered in such case.

This event listener will be registered to the ZoomPanel's JScrollBars (both vertical and horizontal) automatically.

This class is immutable, and thread-safe.

PopupMenuTrigger:

This is an event listener which listens to mouse right clicked event. If the event occurs, popup menu would be shown.

This event listener will be registered to the DiagramView in this component automatically.

This class is immutable, and thread-safe.

ZoomTrigger:

Trigger zoom event for the ZoomPanel if a mouse scroll event is received while Ctrl is pressed.

It implements MouseWheelListener to listen to mouse wheel event, note only when the key "Ctrl" is

pressed the event is processed.

When the mouse wheel is rotated up, zoom in the ZoomPanel;

When the mouse wheel is rotated down, zoom out the ZoomPanel.

The increment of the zoom factor is configured using property “zoom_rotate_increment”.

This class is immutable, and thread-safe.

com.topcoder.gui.diagramviewer.gridrenderers

SimpleGridRenderer

A simple implementation of the GridRender interface. The grid in the active area is painted differently (lighter) than the grid

outside the active area.The grid size can be configured by accessing field gridSize.

Thread safety: this class is mutable and not thread safe.

1.5 Component Exception Definitions

IllegalArgumentException (from java.lang):
This exception is thrown in various methods if the given argument is null. It is also thrown in cases when the passed String parameters to a method are empty Strings. Refer to the documentation in Poseidon for more details.

NOTE: Empty string means string of zero length or string full of white spaces.

IllegalStateException (from java.lang):
This exception is only thrown when setting state of DiagramViewer, if the new state is DiagramState.ADD_NEW_ELEMENT_BY_CLICK or ADD_NEW_ELEMENT_BY_DRAGGING_RECTANGLE.

ConfigurationException (custom):
This exception is thrown to indicate problems in the configuration of this component. It wraps any unexpected exceptions when using ConfigManager and ObjectFactory.

This class is immutable, and thread-safe.

1.6 Thread Safety

This design is not thread safe, since most of the classes have states and are mutable. This will not be a problem in usual GUI application. In a typical Swing application, there will only be a thread access the component's state. This is also known as "The Single-Thread Rule", "Once a Swing component has been realized, all code that might affect or depend on the state of that component should be executed in the event-dispatching thread." It is required for the user to obey this rule to use this component in a Swing application, otherwise, not only this component but also the Java Swing system will have unexpected behavior.

The only thing needs to be noticed is the listeners, since the listener list will be accessed from different thread (main thread and GUI event dispatching thread) it need to be thread safe, we use the listenerList field of JComponent which is thread safe, so it is not a problem here.

However, if this component is used in a multi-threaded environment, e.g., there are several worker threads to complete some kind of task; it will not cause any problems as long as the threads have no access to the state of this component, or all the request is posted via the event-dispatcher. Please refer to the javadoc of SwingUtilities class for how to do this.

If we real want to achieve thread safety of this component, despite this strongly opposed, we only need to make synchronization around the API that has access to the states to achieve thread safety.

2. Environment Requirements

2.1 Environment

- At minimum, Java1.5 is required for compilation and executing test cases.
- Java 1.5 or higher

2.2 TopCoder Software Components

- Base Exception 1.0 was used to provide a base exception for custom exceptions.

- Configuration Manager 2.1.5 was used to provide configuration management.
- Diagram Elements 1.0 was used to provide class Node and NodeContainer.
- Diagram Edges 1.0 was used to provide class Edge.
- UML Diagram Interchange 1.0 was used to provide UML diagram interchange data structure.
- Zoom Panel 1.0 was used to provide zoom feature for the Diagram View.
- Simple Cache 2.0 was used to provide caching of the created Diagram View instances.
- Object Factory 2.0.1 was used to create the simple cache objects and GridRenderer objects.
- `Diagram Interchange 1.0 was used for provide diagram use in classes.`

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.

2.3 Third Party Components

- JUnit : 3.8.2 : http://www.junit.org

NOTE: The default location for 3rd party packages is ../lib relative to this component installation. Setting the ext_libdir property in topcoder_global.properties will overwrite this default location.

3. Installation and Configuration

3.1 Package Name

com.topcoder.gui.diagramviewer

com.topcoder.gui.diagramviewer.event

`com.topcoder.gui.diagramviewer.gridrenderers`

3.2 Configuration Parameters

Note: all parameters in this component is optional, but it is recommended to provide configuration for namespace “com.topcoder.gui.diagramviewer.CloseTabAction”, “com.topcoder.gui.diagramviewer.ZoomInAction” and “com.topcoder.gui.diagramviewer.ZoomOutAction”, these namespaces are used to configure buttons that is expected to be shown only with icon.

Default namespace: com.topcoder.gui.diagramviewer.DiagramViewer

Parameter	Description	Values
object_factory_namespace	the namespace that is used to construct an ObjectFactory	“com.topcoder.gui.diagramviewer.ObjectFactory”

Parameter	Description	Values
max_open_tabs	the max number of open tabs in diagram viewer	number, default to 5
background_grid_visibility	whether the background grid is visible	boolean, default to true
tab_title_displayed_in_full_version	whether the tab title is displayed in full version	boolean, default to false
tab_shortened_title_max_length	the max length of the shortened tab title	integer, 10
diagram_view_background_color	a hex string represents the RGB color value	“FE01A9”
grid_renderer	the name used to create GridRenderer object from ObjectFactory	“gridRenderer”
diagram_view_gap	the gap between the active area and the DiagramView bounds	integer, 50
diagram_view_cache	the name used to create Cache object from ObjectFactory	“viewCache”
zoom_rotate_increment	the increment used to change zoom factor when mouse is rotated while key "Ctrl" is being pressed.	integer, 5
zoom_in_action_namespace	the namespace that is used to create ZoomAction (for zoom in action)	“com.topcoder.gui.diagramviewer.ZoomInAction”
zoom_out_action_namespace	the namespace that is used to create ZoomAction (for zoom out action)	“com.topcoder.gui.diagramviewer.ZoomOutAction”
close_tab_action_namespace	the namespace that is used to create CloseTabAction	“com.topcoder.gui.diagramviewer.CloseTabAction”

Default namespace: com.topcoder.gui.diagramviewer.CloseTabAction

Parameter	Description	Values
name	the name of the button	“close tab”
icon	the icon file path	“com/topcoder/gui/diagramviewer/close.icon”

Default namespace: com.topcoder.gui.diagramviewer.ZoomInAction

Parameter	Description	Values
name	the name of the button	“zoom in”
icon	the icon file path	“com/topcoder/gui/diagramviewer/zoomin.icon”
increment	the increment of the zoom factor	10

Default namespace: com.topcoder.gui.diagramviewer.ZoomOutAction

Parameter	Description	Values
name	the name of the button	“zoom out”
icon	the icon file path	“com/topcoder/gui/diagramviewer/zoomout.icon”
increment	the increment of the zoom factor	-10

3.3 Dependencies Configuration

None needed.

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

1. Provide a configuration file for Configuration Manager and Object Factory.

2. Do necessary work needed by JFC/SWING, such as create JFrame or JPanel, initialize the window manager.

3. Instantiate a DiagramViewer and add it to the window/frame already exist to show it.

4.3 Demo

1. A sample configuration file:

```
<CMConfig>
      <Config name="com.topcoder.gui.diagramviewer.DiagramViewer">

            <Property name="object_factory_namespace">
                  <Value>com.topcoder.gui.diagramviewer.ObjectFactory</Value>
            </Property>

            <Property name="grid_size">
                  <Value>10</Value>
            </Property>

            <Property name="max_open_tabs">
                  <Value>5</Value>
            </Property>


            <Property name="backgound_grid_visiblity">
                  <Value>true</Value>
            </Property>

            <Property name="tab_titile_displayed_in_full_version">
                  <Value>false</Value>
            </Property>

            <Property name="tab_shortened_titile_max_length">
                  <Value>10</Value>
            </Property>

            <Property name="diagram_view_background_color">
            <Value>FE01A9</Value>
            </Property>

            <Property name="grid_renderer">
                  <Value>gridRenderer</Value>
            </Property>

            <Property name="diagram_view_gap">
                  <Value>50</Value>
            </Property>

            <Property name="diagram_view_cache">
                  <Value>viewCache</Value>
```

```
            </Property>

            <Property name="zoom_rotate_increment">
                  <Value>5</Value>
            </Property>

            <Property name="zoom_in_action_namespace">
                  <Value>com.topcoder.gui.diagramviewer.ZoomInAction</Value>
            </Property>

            <Property name="zoom_out_action_namespace">
                  <Value>com.topcoder.gui.diagramviewer.ZoomOutAction</Value>
            </Property>

            <Property name="close_tab_action_namespace">
                  <Value>com.topcoder.gui.diagramviewer.CloseTabAction</Value>
            </Property>
      </Config>


<!-- Configure the Zoom In Button -->
      <Config name="com.topcoder.gui.diagramviewer.ZoomInAction">
            <Property name="name">
                  <Value>zoom in</Value>
            </Property>

            <Property name="icon">
                  <Value>test_files/zoomin.icon</Value>
            </Property>

            <Property name="increment">
                  <Value>10</Value>
            </Property>
      </Config>

<!-- Configure the Zoom Out Button -->
      <Config name="com.topcoder.gui.diagramviewer.ZoomOutAction">
            <Property name="name">
                  <Value>zoom out</Value>
            </Property>

            <Property name="icon">
                  <Value>test_files/zoomout.icon</Value>
            </Property>
            <Property name="increment">
                  <Value>-10</Value>
            </Property>
      </Config>

<!-- Configure the Close Tab Button -->
      <Config name="com.topcoder.gui.diagramviewer.CloseTabAction">
         <Property name="name">
                  <Value>close tab</Value>
            </Property>

            <Property name="icon">
                  <Value>test_files/close.icon</Value>
```

```
            </Property>
      </Config>


<!-- Configure the ObjectFactory -->
      <Config name="com.topcoder.gui.diagramviewer.ObjectFactory">
            <Property name="viewCache">
                  <Property name="type">
                  <Value>com.topcoder.gui.diagramviewer.MockCache</Value>
                  </Property>
            </Property>

            <Property name="gridRenderer">
                  <Property name="type">

      <Value>com.topcoder.gui.diagramviewer.MockGridRenderer</Value>
                  </Property>
            </Property>
      </Config>
</CMConfig>
```

2. Using DiagramViewer

```
// Create a JFrame

JFrame frame = new JFrame();

// Create a DiagramViewer

DiagramViewer viewer = new DiagramViewer();

// Add the viewer to frame

frame.add(viewer);
```

3. Creating a DiagramView for a specified Diagram:

```
//Obtain the Diagram
Diagram diagram = new Diagram();
 diagram.setName("Mock Diagram");
 Dimension dimension = new Dimension();
 dimension.setWidth(500.0);
 dimension.setHeight(500.0);
 Point point = new Point();
 point.setX(5);
 point.setY(5);
 diagram.setPosition(point);
 diagram.setSize(dimension);

 DiagramView view = viewer.createDiagramView(diagram);
```

4. Opening a diagram view tab in DiagramViewer:

```
viewer.openDiagramView(diagram);
```

5. Registering a popup menu for the DiagramView:

```
// Create a JPopupMenu
JPopupMenu popup = new JPopupMenu;
view.setPopupMenu(popup);
// Show the popup menu for the view
//view.getPopupMenu().show(view, 0, 0);
```

5. Show the text input control:

```
// Create something needs to use the text input control
Node node = new ActorNode();
// Get the text input
TextInputBox ti = view.getTextInputBox();
// Register the listener to process text input event
TextInputListener listener = new TextInputListener() {
      public void textEntered(TextInputEvent event) {
            System.out.println("enter text:" + event.getText());
      }
      public void textCancelled(TextInputEvent event) {
            System.out.println("cancel text:" + event.getText());
      }
};
ti.addTextInputListener(listener);
// Show the ti for the component
//ti.show(node, 0, 0);
```

6. Registering for AddNewElementEvent:

```
// Create an implementation of AddNewElementListener
AddNewElementListener newListener = new AddNewElementListener() {
      public void addNewElement(AddNewElementEvent event) {
            DiagramView view = event.getSource();
            Class newElementType = event.getNewElementType();
            // assumes the type is for a concrete Node
            Node node = newElementType.newInstance();
            // Add the new element to view
            view.add(node);
```

```
	}
};
// Register the listener
viewer.addAddNewElementListener(newListener);
```

7. Registering for a ScrollEvent:

```
// Create an implementation of ScrollListener
ScrollListener scrollListener = new ScrollListener() {
	public void diagramViewScrolled(ScrollEvent event) {
		// Obtain the viewport
		JViewport viewport = event.getSource();
		// Obtain the diagram
		Diagram diagram = event.getDiagram();
	}
};
// Register the listener
viewer.addScrollListener(scrollListener);
```

8. Registering for a ZoomEvent:

```
// Create an implementation of ZoomListener
ZoomListener zoomListener = new ZoomListener() {
	public void diagramViewZoomed(ZoomEvent event) {
		// Obtain the ZoomPanel
		ZoomPanel pane = event.getSource();
		// Obtain the new zoom factor
		int factor = event.getNewZoomFactor();
	}
};
// Register the listener
viewer.addZoomListener(zoomListener);
```

9. Showing and hiding the background grid:

```
// Show the background grid
viewer.setBackgroundGridVisible(true);
viewer.revalidate();
// Hide the background grid
```

```
viewer.setBackgroundGridVisible(false);
viewer.revalidate();
```

10. Custom the background grid:

```
// Create an implementation of GridRenderer
GridRenderer myRenderer = new GridRenderer() {
      public void renderGrid(Graphics graphics, DiagramView view){
            // the grid size is 5
            for (int i = 0; i < view.getWidth(); i +=5) {
                  graphics.drawLine(i, 0, i, view.getHeight());
            }
            for (int i = 0; i < view.getHeight(); i +=5) {
                  graphics.drawLine(0, i, view.getWidth(), i);
            }
      }
};
// Register the renderer
viewer.setGridRenderer(myRenderer);
// Repaint
viewer.revalidate();
```

5. Future Enhancements

Provide more user friendly tabbed view.