Event Manager 1.0 Component Specification

1. Design

The Event Manager component provides a general framework for handling events triggered from the GUI. The component handles simple GUI events by notifying the listeners, will handle action events by validating the events, executing the actions and notifying the listeners and will handle undo / redo events.

This design provides Event Manager which can handle action event and GUI event. The manager will allow the user to register/un-register validators for concrete actions or for all actions, action listeners for concrete actions or for all actions and to register listeners for concrete GUI events and for all events.

1.1 Design Patterns

- Observer Pattern is used by GUIEventListener, ActionEventValidator and ActionEventListener to observe the state of each actions.

1.2 Industry Standards

None

1.3 Required Algorithms

1.3.1 Handle Simple Action Event

- ♦ Get the action from the event and get the corresponding validators and listeners by the action.getClass and null.
- ♦ If the validator is not null, for each validator, validate the event with the validator, else set the validationResult to SUCCESSFUL. The result should be the "worst" result returned; validation enum gets progressively worse (from success -> modification -> denied)
- ♦ If validation result is not DENIED, invoke ActionManager.executeAction(action)
- ♦ If the listeners is not null, for each listener, notify the listener by actionPerformed(event, validationResult). Note if same listener contained in both action.getClass and null set, notify it only once.

1.3.2 Handle UndoChanges Event

- ♦ Get the action from the event and get the toBeUndoneActions by ActionManager.getUndoableActionsToBeUndone
- ♦ Remove the actions which is in the list and whose index is bigger than the action
- ♦ Invoke ActionManager.undoActions(action)
- ♦ For each undoneAction, get the listeners of the undoneAction from the actionEventListeners by the undoneAction.getClass and null.
- ♦ For each listener, notify the listener by undoActionPerformed(event, action). Note if same listener contained in both getClass and null set, notify it only once.

1.3.3 Handle RedoChanges Event

- ♦ Get the action from the event and get the toBeRedoneActions by ActionManager.getUndoableActionsToBeRedone

♦ `Remove all the actions which is in the list and whose index is bigger than the action`

♦ `Invoke ActionManager.redoActions(action)`

♦ `For each redoneAction, get the listeners of the redoneAction from the actionEventListeners by the redoneAction.getClass and null.`

♦ `For each listener, notify the listener by redoActionPerformed(event, action). Note if same listener contained in both getClass and null set, notify it only once.`

1.4 Component Class Overview

EventManager:

This class is the main class of this component. The manager will be responsible for handling action events and GUI events. And user can add/remove validator & listener for specified class of action/event.

- For the simple action event, it will validate it by the registered validator and execute it according to the validation. And notify the registered action event listeners.
- For the undo/redo changes event, it will simply redo/undo it and notify every undone/redone actions by their registered listeners.

ActionEvent:

The action event extends EventObject and has a reference to the action to be executed. The action will be executed by the action manager.

UndoChangesEvent:

This class extends from the ActionEvent. The action inside must be an UndoableAction. The action will be undone by the action manager.

RedoChangesEvent:

This class extends from the ActionEvent. The action inside must be an UndoableAction. The action will be redone by the action manager.

ActionEventValidator:

This interface specifies the contract for implementations of an action event validator. It represents the validator invoked by the manager in order to validate an action event.

EventValidation:

This enumeration represents the validation result type.

ActionEventListener:

This interface specifies the contract for implementations of an action event listener. It represents the listener notified by the event manager about action events and about the undo / redo events. It provides a method to be invoked about the execution of an action, using the action event and the validation result. It will also provide two methods to be used when an undo / redo event occured. These methods will be invoked for each undoable action that is undone / redone.

GUIEventListener:

This interface specifies the contract for implementations of an GUI event listener. It represents the listener notified by the event manager about simple GUI events.

1.5 Usage of Exceptions

ActionExecutionException:

This exception will be re-thrown by handleSimpleActionEvent if it is thrown by

ActionManager#executeAction, and it will be re-thrown by handleActionEvent if it is thrown by handleSimpleActionEvent. It will expose to the caller of handleActionEvent methods.

IllegalArgumentException:
This exception will be thrown by the constructors of EventManager, ActionEvent, RedoChangesEvent and UndoChangesEvent if any argument is null. It will also be thrown by the handleActionEvent, handleGUIEvent and the add/remove listener/validator methods if any argument is null. And it will also be thrown by the add/remove listener/validator methods if given class is not the expected type.

1.6 Thread Safety

This component is not thread-safe since the Map is used by the EventManager. The other classes are thread-safe by being immutable.

This component is considered to be used in single thread. If an application does intend to use this component in a multi-threaded environment, then it should ensure the add/remove validator/listener methods and handleActionEvent/handleGUIEvent methods are synchronized externally.

2. Environment Requirements

2.1 Environment

- Development language: Java1.5
- Compile target: Java1.5

2.2 TopCoder Software Components

- Action Manager 1.0 is used to execute the action and get the to be undone/redone event.

2.3 Third Party Components

None

3. Installation and Configuration

3.1 Package Name

com.topcoder.util.eventmanager

3.2 Configuration Parameters

None

3.3 Dependencies Configuration

None

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

Nothing special required

4.3 Demo

4.3.1 *Simple Demo for Action Event*

```
        // The ActionManager is created by the user
        actionManager = new MockActionManager();

        List<UndoableAction> actionsTobeUndones = new
ArrayList<UndoableAction>();
        actionsTobeUndones.add(undoableAction1);
        actionsTobeUndones.add(undoableAction2);
        actionsTobeUndones.add(undoableAction3);
        // Mock up the actionManager
        actionManager.setUndoableActionsToBeUndone(actionsTobeUndones);
        actionManager.setUndoableActionsToBeRedone(actionsTobeUndones);
        // The EventManager is created by the user
        eventManager = new EventManager(actionManager);

        // Register the validators to UndoableAction.class
        eventManager.addEventValidator(successEventValidator,
UndoableAction.class);

        // Register the validators to all Action class
        eventManager.addEventValidator(modifiedEventValidator);

        // Register the validators to simple Action Class
        eventManager.addEventValidator(deniedEventValidator, Action.class);

        // The following listeners implement ActionEventListener
        // Register the actionEventListener1 to MockUndoableAction class
        eventManager.addActionEventListener(actionEventListener1,
MockUndoableAction.class);

        // Register the actionEventListener2 to all the action events
        eventManager.addActionEventListener(actionEventListener2);

        // Register the actionEventListener3 to simple Action Class
        eventManager.addActionEventListener(actionEventListener3,
Action.class);

        // Create the action event, and handle it
        ActionEvent actionEvent = new ActionEvent(undoableAction1, new
String());
        // Handle the event, actionEventListener1 and actionEventListener2
should be notified
        // If the validation by the validator is not DENIED, the action will
be executed.
        eventManager.handleActionEvent(actionEvent);

        // Create an undo changes event, and handle it
        UndoChangesEvent undoChangesEvent = new
UndoChangesEvent(undoableAction2, new String());
        // Handle the event, actionEventListener1 and actionEventListener2
should be notified
        eventManager.handleActionEvent(undoChangesEvent);

        // Create a redo changes event, and handle it
        RedoChangesEvent redoChangesEvent = new
RedoChangesEvent(undoableAction2, new String());
        // Handle the event, actionEventListener1 and actionEventListener2
```

```
should be notified
       eventManager.handleActionEvent(redoChangesEvent);
```

4.3.2 Simple Demo for GUI Event

```
      // The event is triggered from the GUI, it will be passed to the event
manager to be handled.
       RedoChangesEvent guiEvent = new RedoChangesEvent(undoableAction1, new
Object());

       // Handle the event, no listeners will be notified
       eventManager.handleGUIEvent(guiEvent);

       // Register the following listeners which implements GUIEventListener
       // Register the gUIEventListener1 to GUI event
       eventManager.addGUIEventListener(gUIEventListener1,
RedoChangesEvent.class);

       // Register the gUIEventListener2 to all events
       eventManager.addGUIEventListener(gUIEventListener2);

       // Handle the event, gUIEventListener1 and gUIEventListener2 will be
notified
       eventManager.handleGUIEvent(guiEvent);
```

5. Future Enhancements

None