Handling Keyboard Events

An Oracle White Paper
December 2005

ORACLE

Handling Keyboard Events

Handling Keyboard Events

INTRODUCTION

A common feature in desktop application development is to provide keyboard shortcuts for specific application functionality to make the end user more productive when working with an application. Oracle ADF Swing, the Swing development framework within Oracle JDeveloper, supports the definition of keyboard shortcuts through default Swing mnemonics and through the Swing component's InputMap and ActionMap objects.

This whitepaper explains how application developers can define custom keyboard shortcuts for functionality and UI components that are exposed in ADF Swing applications. Swing components also provide a set of default keyboard mappings and shortcuts that are worth examining[1] before building custom keyboard handling functionality in an application.

ORACLE ADF SWING

Oracle JDeveloper contains Oracle ADF Swing, an ease of use Swing development framework that provides declarative Swing component binding to the Oracle Application development Framework (Oracle ADF). Using Oracle ADF in combination with Oracle ADF Swing, application developers will find it easy to develop enterprise-class Swing applications that use Oracle ADF Business Components, OracleAS TopLink, Enterprise JavaBeans (EJB), web services or Plain Old Java Objects (POJOs) as their database access and persistence layer.

All application development in Oracle ADF Swing is either declarative or requires Swing and Java development skills.

MNEMONICS

Mnemonics are keyboard shortcuts, which are defined for Swing UI components as a combination of "Alt+key" for the component to request focus.

UI components that have a mnemonic defined typically indicate this in the associated label or prompt with an underlined character. The underlined character designates the key to press in combination with the "Alt" key. For this to work, field labels need to be created and associated with the UI component. Oracle

[1] http://java.sun.com/j2se/1.4.2/docs/api/javax/swing/doc-files/Key-Index.html

JDeveloper supports, declaratively in its Property Inspector (Ctrl+Shift+I), both the label-to-field association and the definition of the mnemonic.

All settings performed in the Property Inspector are added as JavaBean method calls to the Java file that defines the Swing component. This also means that Swing developers can programmatically change the mnemonic key that is assigned to a UI component. Though its possible to set mnemonics dynamically, it is not an action we often see performed in Swing applications.

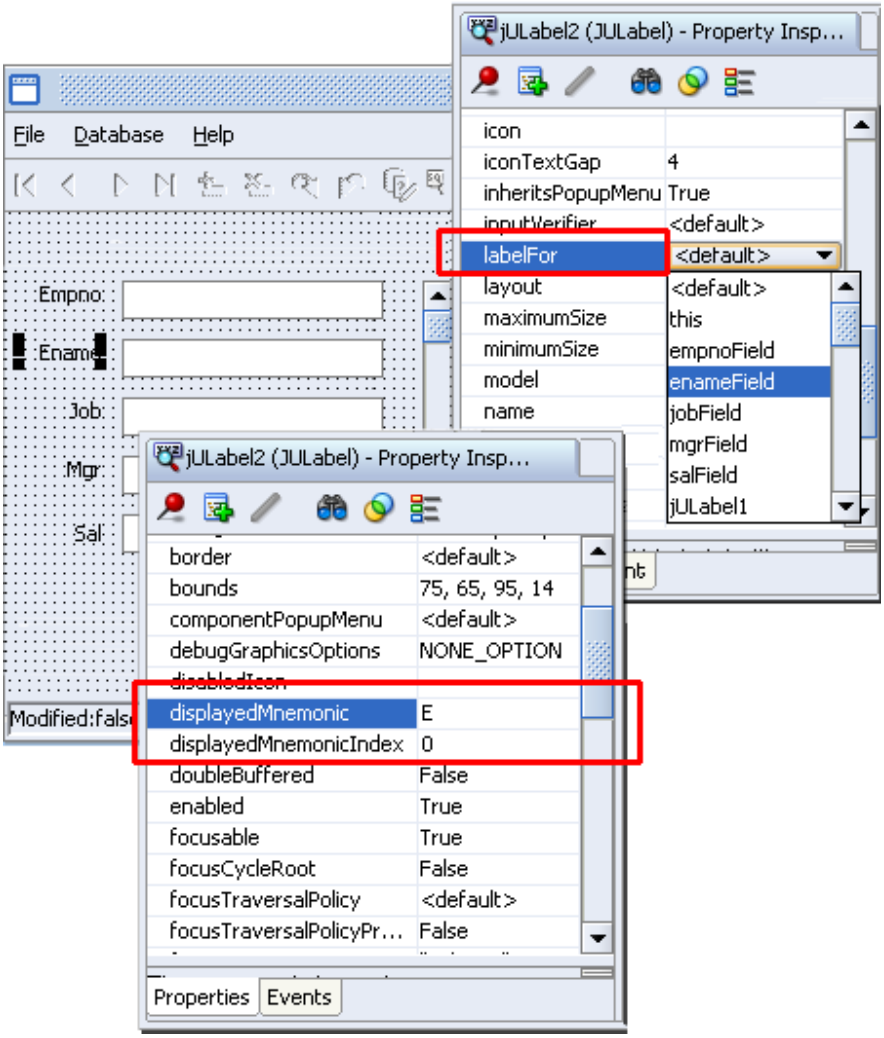

Figure 1: Creating Mnemonics for a Swing JTextField

KEY LISTENERS

Event listeners respond to specific JavaBean events. KeyEvent listeners in Swing are registered on UI components to respond to a specific keyboard action.

In Oracle JDeveloper, event listeners can be implemented manually in the source editor or declaratively, using the Property Inspector.

Declaratively Assigning Key Listeners

To declaratively define a key event listener on a Swing UI component, select the component in the Swing Visual Editor and open the Property Inspector (Ctrl+Shift+I).

Switch to the event properties by selecting the **Events** tab at the bottom of the Property Inspector.

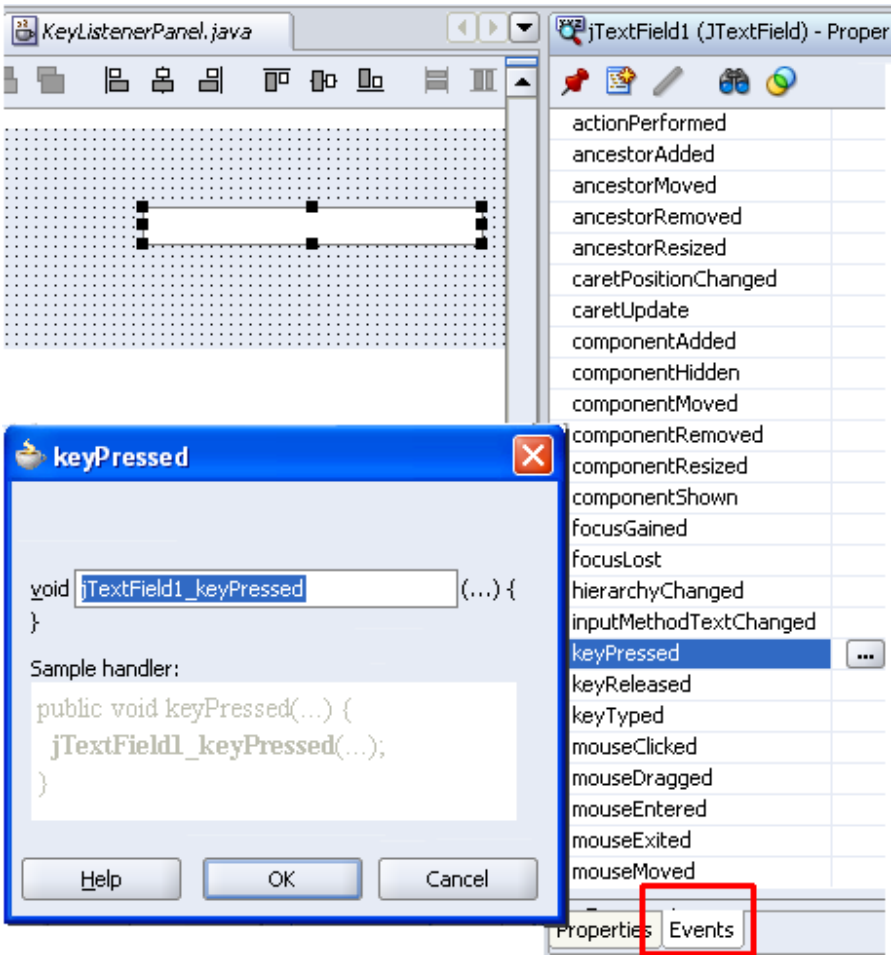

Figure 2: Creating Key Event Handlers using the Property Inspector in Oracle JDeveloper

Clicking the ... button in the Property Inspector for one of the key events opens a dialog to specify the name of the method stub to create for the event handler. After clicking the OK button in this dialog, Oracle JDeveloper adds the appropriate Swing listener to the UI component and points it to the generated stub method to execute whenever the event occurs.

Oracle JDeveloper opens the source editor on the generated method stub to allow the application developer to customize the keyboard event.

The following code added to the stub in the Oracle ADF Swing source editor prints a message whenever the Enter key is pressed while the JTextField component has focus.

```
private void jTextField1_keyPressed(KeyEvent e) {
   if (e.getKeyCode() == KeyEvent.VK_ENTER){
      System.out.println("Enter pressed");
   }
}
```

Assigning Key Listeners Manually

Instead of selectively adding a specific keyboard listener to a Swing component, as it is done with the declarative approach explained above, you can alternatively implement all KeyEvent listeners at once using the source editor in Oracle JDeveloper.

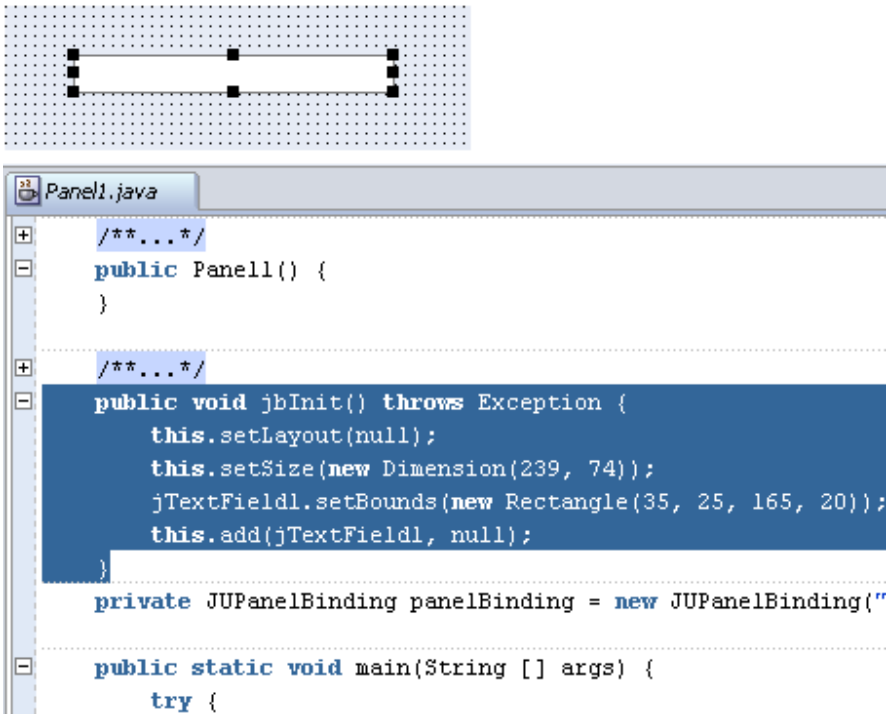

Figure 3: JTextField as it Appears in the Swing Visual Editor and Source Editor

A JTextField in Oracle ADF Swing is implemented in the panel source code as shown below:

To add a KeyBoard event listener to the JTextField, add the following code in the source editor.

```
jTextField1.addKeyListener(new KeyListener(){

});
```

This code is not functional yet because it does not implement all the methods required by the KeyListener interface.

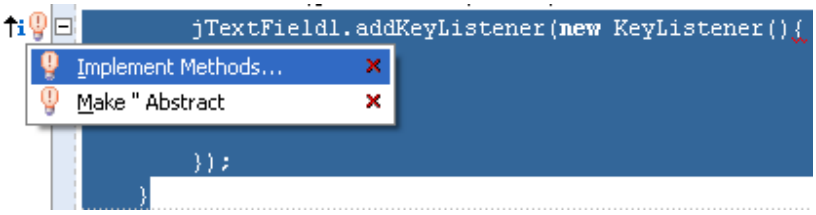

Figure 4: Oracle JDeveloper Quick Fixes Assistant in Source Editor

Oracle JDeveloper helps you complete this task with a Quick-Fix assistant that displays as a light bulb icon in the source editor margin next to the listener definition. The upwards-pointing arrow icon indicates that the code line needs to implement methods that are required by the KeyListener interface.

Selecting the **Implement Methods** option creates the following addition to the code:

```
jTextField1.addKeyListener(new KeyListener(){
   public void keyTyped(KeyEvent e) {}
   public void keyPressed(KeyEvent e){}
   public void keyReleased(KeyEvent e) {}
});
```

The only work left is to provide the logic that specifies how the keyboard event should be handled. For example, as previously shown in the section above:

```
public void keyPressed(KeyEvent e){
if (e.getKeyCode() == KeyEvent.VK_ENTER){
    System.out.println("Enter pressed");
}
```

KEYBOARD BINDING WITH INPUT MAPS AND ACTION MAPS

The InputMap class provides a binding between a keystroke and an associated action on an object. The action is defined in the component ActionMap class, an internal table that maps a key to an implementation of the *javax.swing.Action* interface.

The easiest way to build an implementation of the Action interface is to subclass the abstract *javax.swing.AbstractAction* class, and override its *actionPerformed()* method.

Depending on the need to reuse the defined action, the Action class implementation can be defined either as an inner class or as an external class,.

Most components in Swing, like JPanel, JButton and JTextField, are descendants of the *javax.swing.JComponent* class, which exposes InputMap and ActionMap for registering keyboard action mappings through getter and setter methods. This functionality was added to Swing in J2SE 1.3.

The JComponent class also contains the static constants WHEN_IN_FOCUSED _WINDOW, WHEN_ANCESTOR_OF_FOCUSED_COMPONENT and WHEN_ FOCUSED to determine the scope of a keyboard mapping on a component. The constant WHEN_IN_ FOCUSED_WINDOW provides the broadest focus for a keyboard binding and will execute when the mapping is defined on a component that is within the range of the focused Swing window.

The component keyboard mappings defined using InputMap and ActionMap will consume keyboard events that have not already been handled. A key listener defined on a component that consumes the same keyboard event will have precedence over InputMap handling.

Using InputMap, Swing checks if a keyboard event is registered with a component and if a value other than null is returned, checks the component ActionMap for the action to perform.

ADF Swing InputMap and ActionMap Example:

The following code example defines a keyboard mapping for the F8 and F11 keys in the Oracle ADF Swing helper form to execute "enter query" and "execute query" mode functionality. Oracle ADF Swing in Oracle JDeveloper 10.1.3 automatically creates the helper form class as an instance of JPanel when dragging a collection, for example by dragging an ADF Business Components view object, as an input form from the Data Control Palette to a Swing panel or frame.

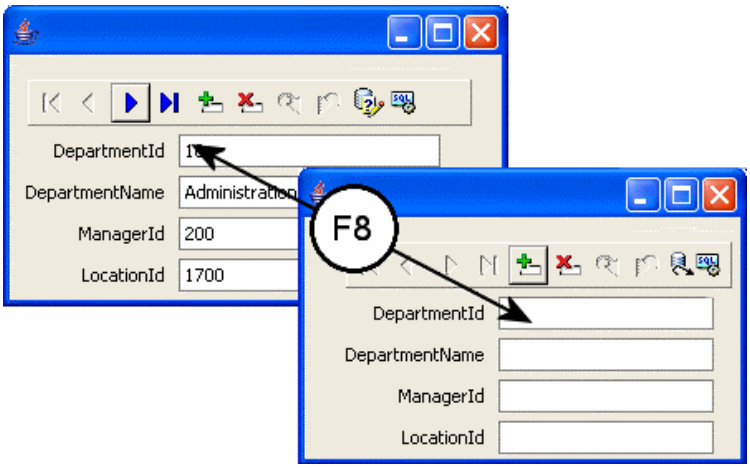

Figure 5: Binding F8 to "Find Mode" in Oracle ADF Swing

In the code example below, the dropped Collection was DepartmentsView, generating the PanelDepartmentsView1Helper class.

```
public class PanelDepartmentsView1Helper extends JPanel
implements JUPanel {
…
class ExecuteDepartmentsAction extends AbstractAction {
  public void actionPerformed (ActionEvent e){
    //obtain execute query functionality from the associated
    //toolbar (navBar)

navBar.getButton(JUNavigationBar.BUTTON_EXECUTE).doClick();
  }
}
class EnterQueryDepartmentsAction extends AbstractAction {
   public void actionPerformed (ActionEvent e){
    //obtain enter query functionality from the associated
    //toolbar (navBar)

     navBar.getButton(JUNavigationBar.BUTTON_FIND).doClick();
   }
}
//within the jbInit() method
//Define EnterQuery on F8
this.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW)
  .put(KeyStroke.getKeyStroke("F8"),"QueryDepartment");
this.getActionMap().put("QueryDepartment",new
  EnterQueryDepartmentsAction());
//Define Execute Departments on F11
this.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW)
   .put(KeyStroke.getKeyStroke("F11"),"ExecuteDepartment");
this.getActionMap().put("ExecuteDepartment",new
   ExecuteDepartmentsAction());
…
}
```

If you want to disable an existing keyboard mapping in an application, you do this by registering the action "none" for it. By convention "none" never has an action associated with it in the ActionMap.

SUMMARY

This whitepaper introduces different strategies for assigning functionality to keyboard shortcuts in Oracle ADF Swing applications. The use of mnemonics is the easiest way to implement keyboard shortcuts. A defined mnemonic is bound to a specific component and does not support executing custom code. Using a key listener or keyboard maps is more flexible and supports custom coding.

Handling Keyboard Events
February 2006
Author: Frank Nimphius
Contributing Authors: [OPTIONAL]

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
oracle.com