

---

## *Classic Key Events*

---

*NOTE* This is the key binding discussion from the first edition of *Java Swing*. It has been preserved for anyone who still needs to use an SDK prior to 1.3. SDK 1.3 introduced a new mechanism, described in our second edition.

---

### *Keyboard Actions*

*public void registerKeyboardAction(ActionListener anAction, String aCommand, KeyStroke aKeyStroke, int aCondition)*

*public void registerKeyboardAction(ActionListener anAction, KeyStroke aKeyStroke, int aCondition)*

These methods register a specific keyboard action with the component. When the keystroke `aKeyStroke` occurs under the appropriate conditions, `JComponent` invokes the `actionPerformed()` method of the object implementing `anAction`. If the programmer desires, the action command can be set to `aCommand`. The conditions involved are listed in Table 3-1.

*public void unregisterKeyboardAction(KeyStroke aKeyStroke)*

Unregisters a keyboard action from the component.

*public int getConditionForKeyStroke(KeyStroke aKeyStroke)*

Returns the conditions defined for the keyboard action triggered by `aKeyStroke`.

*public ActionListener getActionForKeyStroke(KeyStroke aKeyStroke)*

Returns the `Action` that is registered to be triggered by `aKeyStroke`.

*public void resetKeyboardActions()*

Clears all keyboard actions for the component.

*protected void processComponentKeyEvent(KeyEvent e)*

This protected method is called if there are no keyboard actions matching the keystroke directed at the component. The method currently does nothing; you can override it in your own components to perform component-specific keyboard handling.

## Keyboard Events

Swing components can be programmed to trigger various actions when certain keystrokes occur. For example, Swing components automatically handle focus-related keyboard events. The default focus mechanism watches only for **Tab** and **Shift-Tab** keystrokes, altering the focus and consuming the keystrokes when detected. If the focus mechanism does not know how to handle a keystroke, it checks to see whether the `processComponentKeyEvent()` method can consume it. This method currently does nothing. However, you can override it in a subclass if you want to react to a keystroke in your own way. If neither of these succeeds in consuming the key event, the `JComponent` class checks to see if a *keyboard action* has been registered for that keystroke. A keyboard action binds an action and a keystroke to a specific component.

You can associate keyboard actions to a component through one of two `registerKeyboardAction()` methods:

```
void registerKeyboardAction(ActionListener anAction, String aCommand,
                           KeyStroke aKeyStroke, int aCondition)
void registerKeyboardAction(ActionListener anAction, KeyStroke aKeyStroke,
                           int aCondition)
```

In order to use these methods, you need a couple of things: a `KeyStroke` object that represents the keystroke combination you wish to monitor, and an `Action` that occurs when that keystroke has taken place. (Actions are covered earlier in the chapter.) If you want the keyboard event to insert a specific string as the event's action command, you can specify that as well. A final parameter to the `registerKeyboardAction()` method places restrictions on when the action is fired. Table 3-1 shows the constants for the final parameter.

Table 3-1. Constants for Keystroke Registration

Constant	Description
<code>WHEN_FOCUSED</code>	The <code>Action</code> takes place only when the target component has the focus.
<code>WHEN_IN_FOCUSED_WINDOW</code>	The <code>Action</code> takes place when the target component has the focus or resides in a container that has the focus.
<code>WHEN_ANCESTOR_OF_FOCUSED_WINDOW</code>	The <code>Action</code> takes place when the target component has the focus, or is the ancestor of the component that currently has the focus.

The latter two constants come in handy in different situations. `JComponent.WHEN_IN_FOCUSED_WINDOW` can be used in a situation where you want to define a keystroke that works on a specific component in a container, but functions even when the parent container has the focus. For example, a dialog could have a save button that captures the letter “S” in conjunction with a modifier key (**Alt**, **Ctrl**, or **Shift**). It could register this keyboard action with the button to perform a save when the “S” key was pressed. The keyboard action would work even if the button didn’t explicitly have the focus—as long as the parent dialog did. On the other hand, if you want to define a keyboard event that is standardized throughout a container and its components, register a keyboard event at the container level and use the `JComponent.WHEN_ANCESTOR_OF_FOCUSED_WINDOW` option. This allows keyboard events to be recognized at the container level, even if the descendant has the focus.

You can use the `registeredKeyStrokes` property of `JComponent` to get access to all the keystrokes registered for a component. The `getRegisteredKeyStrokes()` method returns this information as an array of `KeyStroke` objects. If you want to see what condition is tied to any particular keystroke, you can use the `getConditionForKeyStroke()` method. If you want to obtain the action associated with the keystroke, use `getActionForKeyStroke()`. In both cases, you will have to specify the target keystroke as the search key.