```
            hookResult = 1
            break
         case 30:
            Print "Up arrow key pressed."
            hookResult = 1
            break
         case 31:
            Print "Down arrow key pressed."
            hookResult = 1
            break
         default:
            // The keyText field requires Igor Pro 7 or later
            // See Keyboard Events on page IV-300
            Printf "Key pressed: %s\r", s.keyText
            break
      endswitch
      break
   endswitch

   return hookResult // If non-zero, we handled event and Igor will ignore it.
End

Function DemoWindowHook()
   DoWindow/F DemoGraph    // Does graph exist?
   if (V_flag == 0)
      Display /N=DemoGraph // Create graph
      SetWindow DemoGraph, hook(MyHook)=MyWindowHook  // Install window hook
   endif
End
```

The window hook function receives a WMWinHookStruct structure as a parameter. WMWinHookStruct is a built-in structure that contains all of the information you might need to respond to an event. One of its fields, the eventCode field, specifies what kind of event occurred.

If your hook function returns 1, this tells Igor that you handled the event and Igor does not handle it. If your hook function returns 0, this tells Igor that you did not handle the event, so Igor does handle it.

This example uses a named window hook. In this case the name is MyHook. More than one procedure file can install a hook on a given window. The purpose of the name is to allow a package to install and remove its own hook function without disturbing the hook functions of other packages. Choose a distinct hook function name that is unlikely to conflict with other hook names.

Earlier versions of Igor supported only one unnamed hook function. This meant that only one package could hook any particular window. Unnamed hook functions are still supported for backward compatibility but new code should always use named hook functions.

Window Hooks and Subwindows

Except for the modified event (see **Modified Events** on page IV-299), Igor calls window hook functions for top-level windows only, not for subwindows. If you want to hook a subwindow, you must set the hook on the top-level window. In the hook function, test to see if the subwindow is active. For example, this code, at the start of a window hook function, insures that the hook runs only if a subwindow named G0 is active.

```
GetWindow $s.winName activeSW
String activeSubwindow = S_value
if (CmpStr(activeSubwindow,"G0") != 0)
   return 0
endif
```

Exterior panels (see **Exterior Control Panels** on page III-443) are top-level windows even though they are subwindows. To hook an exterior subwindow, you must install the hook on the exterior panel using subwindow syntax.