SpyWorks demonstration controls.

The following is a brief description of the properties and use of the SpyWorks demonstration controls included on this disk. Only 32 bit controls are included here - 16 bit OLE Controls and VBX versions of these controls are also available.

Note: These controls are provided for educational use and experimentation only. No technical support is available for this demonstration control at this time, though we may be able to provide limited online support at some future time (Send an e-mail message to 74431,3534 on CompuServe or 74431.3534@compuserve.com to receive updated information). This is especially important because the SpyWorks technology can be extremely dangerous to use—it is safe and reliable once your code is correct, but it is common during the debugging process to crash not only your application, but others as well (not to mention, the entire system).

The control descriptions here are brief, but you will find several examples in the SpyWorks subdirectory that illustrate use of these controls. Extensive examples for the dwcbk32d.ocx and dwsbc32d.ocx controls can be found in the Visual Basic Programmer's Guide to the Win32 API.

dwcbk32d.ocx:

This is the SpyWorks callback control. It provides a pool of function addresses that can be passed to Windows or DLL's that use callbacks. When Windows or the DLL calls the provided function address, and event is triggered in the control.

Properties

(About) :When the "..." button in the property window is clicked, it will display information about the version and capabilities of the dwcbk32d.ocx custom control.

Convention: All API callbacks use the ‘Pascal’ or stdCall calling convention.

EventTrigger: Set to Immediate for use with API enumeration functions. Posted causes the dwcbk32d.ocx control to defer the actual event trigger by way of a posted message. This allows the control to work with some interrupt level callbacks. In this case the value of the PostedReturn property will be returned to windows.

Type: Each callback type has a specific set of parameter types and sizes and triggers a corresponding event. Refer to your Windows API reference for information on callback functions and their parameters.

Type32Enabled: When True, the Type will be set by the Type32 property instead of the Type property. This allows you to specify different callbacks for 16- and 32-bit versions of the control.

ProcAddress: When the Type property is not zero (disabled), this property returns the address of a callback function that can be passed to an enumeration function.

Event Example:

EnumWindows

Type property = 6—EnumWindows

Description

The ProcAddress property provides the address for a function suitable for use with the

EnumWindows and other windows enumeration API functions. This callback function is called for each window that is enumerated.

Visual Basic Event Triggered:

EnumWindows(hWnd&, lParam&, retval&)

hWnd: A handle to the window being enumerated.
lpParam: This is the same value passed by your program as the lParam parameter to the EnumWindows function. This is typically used to pass information to the callback function.
retval: Set this value to zero to stop the enumeration.

dwSbc32d.ocx

dwsbc32d.ocx is a custom control that allows Visual Basic applications to subclass windows in their own or other applications. This control is a demonstration version of the dwsbc32.ocx subclassing custom control provided with Desaware's SpyWorks program.

Using dwsbc32d.ocx

The dwsbc32d.ocx custom control is added into your Visual Basic controls' toolbox using the File-Add command in the same way as any other Visual Basic custom control. Like the timer control, the dwcbk32d.ocx custom control is not displayed at runtime.

In general, using the dwsbc32d.ocx control requires three steps:
· Select the window to subclass either at design time (using the ctlParam property) or at runtime (by setting the HwndParam or AddHwnd property).
· Select the messages to detect using the Messages property.
· Add Code to the WndMessage or WndMessageX event to execute when the message is detected.
Use the Type property to specify if you wish to intercept the message before the default processing for the window, after the default processing for the window, or simply wish it to be posted as a notification.
When a message is detected by the dwsbc32d.ocx control, the WndMessage or WndMessageX events are triggered (depending on whether the message comes from the same application or a foreign application). This event contains all of the standard window function parameters (hWnd, msg, wparam, lparam), and two additional parameters: nodef and retval, which are described under the description of the Type property.
A single dwsbc32d.ocx control can subclass any number of windows. A primary window to subclass may be specified using the CtlParam or HwndParam properties. Additional windows can be added to an internal subclassing array using the AddHwnd property. All windows subclassed by a particular control share the same list of messages to detect.

There are many operations that cannot be performed safely during processing of messages, especially when the Type property is set to zero or one. You should especially avoid use of message boxes or breakpoints. For debugging, use the debug.print statement to obtain information on what is going on during your message routines.

Most Frequently Used dwsbc32d.ocx Properties

(About):When the "..." button in the property window is clicked, it will display information about the version and capabilities of the DWSBC32D.OCX custom control.

AddHwnd: Assigns a window handle to this property to add the window to an internal array of windows to subclass.

ClearMessage: Assigns a message number to this property to remove the message from the list of messages to detect.

CtlParam: Sets this property to the name of a form or control to subclass.

HookCount: Reads this property to determine how many windows are being subclassed using the internal subclassing array.

HwndParam: Sets this property at runtime to the main window you wish to subclass if you do not use the CtlParam parameter to choose the window by name. This window is in addition to any windows specified by the internal subclassing array.

MessageArray: This property array contains all of the messages currently being detected.

MessageCount: This property contains the number of messages in the MessageArray.

Messages: This design time property is a placeholder. Click on the property select button in the property window to bring up the message selection dialog box.

PostEvent: Sets this property to a value to cause a message to be posted to the control. This will cause a DelayedEvent event to be triggered during the course of normal Windows processing with the long value set into this property as a parameter. This allows you to "do something later" without using a timer.

RegMessage1 through RegMessage5: These properties can be set to names of registered messages that will be detected.

RegMessageNum: This property array (with five entries) allows you to retrieve the numbers for registered messages RegMessage1 through RegMessage5 without calling the RegisterWindowMessage API function.

RemoveHwnd: Removes a window handle from the internal subclassing array. This stops subclassing on the specified window. To stop subclassing on the primary window (specified using the HwndParam or CtlParam properties), set the property to zero or the empty string respectively.

Type: Pre-Default: Intercept the message before sending it to the default window function. You can set the nodef parameter in the WndMessage or WndMessageX event to True to prevent default processing from taking place. Set the RetVal parameter in these events to return a value to the calling routine.PostDefault: Intercept the message after default processing. The RetVal parameter contains the value returned by the default window function and may be modified.Posted: The WndMessage or WndMessageX events are triggered after the messagehas completed processing. The event is triggered during the normal course of message processing. Changing parameters has no effect. Any messages that use memory pointers may not refer to valid data at this time. This is the safest subclassing mode.

UseOnlyXEvent All messages trigger the WndMessageX event regardless of whether they are in process or out of process.

dwShk32d.ocx

dwShk32d.ocx is a powerful Windows hook and keyboard hook control. Hooks differ from subclassing in that they have the ability to intercept messages on a more global basis, for example: for all controls on a form, in a process or even for every window in the system.
This control differentiates between standard hook types and keyboard hook types, allowing you to enable either or both types of hooks.

Using dwshk32d.ocx for keyboard hooks.
- Use the KeyboardHook property to enable the keyboard hook and determine the scope of the hook.
- Use the Keys property to select the keys that you wish to detect.
- Add code to the KeyUpHook and KeyDownHook events to process the intercepted keystrokes.

Using dwshk32d.ocx for other hooks.
- Use the HookEnabled property to enable the hooks.
- Use the Monitor property to determine the scope of the messages detected.
- Use the Messages property to select the messages to detect.
- Use the HookType property to choose the type of hook.
- Add code to the appropriate hook event to process the detected messages.

dwshk32d.ocx properties

(About):When the "..." button in the property window is clicked, it will display information about the capabilities of the DWSHK32D.OCX custom control.

ClearKey: Call this method with a key value to remove a key from the list of keys to detect using a keyboard hook.
A key value in dwshk.ocx is represented by a long value. The low 16 bits corresponds to the virtual key number (which is identical to the KeyCode parameter to the VB KeyDown and KeyUp events). The high 16 bits determine the requested state of the SHIFT, CONTROL and ALT keys where bit 0 corresponds to the state of the SHIFT key, bit 1 corresponds to the state of the CTRL key and bit 2 corresponds to the state of the ALT key. This matches the shift parameter to the VB KeyDown and KeyUp events.

HookEnabled: Set to True to enable the hooks portion of this control (not including keyboard hooks).

HookType: Set to the type of hook to enable. Refer to your Windows API reference for information under the SetWindowsHookEx function for information on the different types of hooks. It is important to choose the correct hook for an application, as not every hook type intercepts all messages. Each hook type has a corresponding event (though some hook types share events).

Monitor: Determines the scope of the hook. You can monitor all messages for a form, for the current or specified process or thread, or for the entire system.

Notify: Determines if a hook event is triggered immediately or is deferred. When triggered immediately, it may be possible to modify or discard the message (depending on the type of hook).

KeyArray: A property array that allows you to read a list of the key values being detected with keyboard hooks.

KeyboardHook: Use to enable the keyboard hook capability of this control. Keys can be detected for the current task, any specified task or thread, or the entire system.

KeyCount: A property containing the number of key values being detected.

Keys: This design time property is a placeholder. Click on the property select button in the property window to bring up the key selection dialog box.

KeyboardNotify: Determines if a keyboard event is triggered immediately or is deferred. When triggered immediately, it is possible to modify or discard the keystroke.

dwshk32d.ocx events

KeyDownHook, KeyUpHook
The KeyCode parameter for these events contains the key code as defined by the Visual Basic KeyUp and KeyDown events. The ShiftState parameter also corresponds to these events. Repetitions is the repeat count for the key.
The Keystate paramter is as follows:

Bit 0 - 7	the hardware dependent scan code.
Bit 8 = 1	if this is an extended key (function key or on the numeric keypad)
Bit 9 = 1	if this key is being "peeked" (i.e. this event was not removed from the system queue). This is only possible if the KeyViewPeeked property is set to True.
Bit 13 = 1	if the ALT key is down
Bit 14 = 1	if the key was previously down, 0 if it was up.
Bit 15 = 1	if the key is being released, 0 if pressed.

If the KeyboardNotify property is 0, you can set the KeyCode parameter to zero and the discard parameter to True to throw away the keystroke.

Refer to your Win32 API reference for a description of the various other hook events.

dwEasy32.ocx

The dwEasy32.ocx control provides a prepackaged set of subclassing based capabilities. Unless otherwise specified, each feature is applied to the form or control that contains a particular dwEasy32.ocx control.

Mouse Tracking:
Use the MouseTransit property to enable mouse tracking - the detection of mouse enter and mouse exit events for windows based controls. The MouseEnter and MouseExit events are only enabled when the MouseTransit property is set to True. The TransitTag property allows you to quickly read the Tag property of the control that has most recently been entered or exited.

Menu Selection:
The MenuSelect event is triggered any time a menu is highlighted for the parent form. This is ideal for implementing status bars based on menu selection state. If the MDIMenuDetect property is set to True, and the control is placed on a picture control (or other container) that is on an MDI form, or an MDI child form, the MDIMenuSelect event will be triggered for menu selection events on the MDI form's menu. These events provide the text information from the menu, the position of the menu, and a set of flags that match the MF_?? flags in the Win32 API. The System Command event is triggered when system menu commands are executed, making it possible to customize the system menu or discard system menu events if you wish.

Update Area:
During a paint event of a form or control containing a dweasy32.ocx control, you can read the UpdateLeft, UpdateTop, UpdateRight and UpdateBottom properties to determine the area of the control that actually needs to be redrawn. This can be used to improve performance for applications that use the Paint event to create complex drawings. These properties are specified in pixels.

Tiny Captions:
This capability allows you a great deal of control over the look of the caption of the form or picture control containing the dweasy32d.ocx control. The CaptionStyle property allows you to specify a resized caption or a "rolllup" style caption. Set the CaptionHeight property to 0 to 200 indicating the % of normal size for the caption if the CaptionStyle property is non-zero.
The ForceActive, ForceTitle, ForceCtlBox and ForceOutline properties allow you to control the

appearance of a caption created on a picture control (which does not have properties to control attributes such as the text of the caption).
The RolledUp property is used when the CaptionStyle property is set to 2 (roll-up) to determine or control whether the window is rolled up or visible.

ScrollBars:
Use the ScrollBars property to turn on the horizontal or vertical scrollbars for a form or picture control. The ScrollWindow property is used to turn on virtual forms - the scrollbars automatically scroll controls on the form. The ScrollViewport also shifts the underlying coordinate system for the form or control, allowing drawing operations to work correctly on the scrolled area. The VLargeChange, VSmallChange, VMax, VMin, VValue, HLargeChange, HSmallChange, HMax, HMin and HValue properties correspond to the Visual Basic ScrollBar control LargeChange, SmallChange, Min, Max and Value properties for the vertical and horizontal scrollbars respectively.