CHILD WINDOW CONTROLS

Child window controls are a powerful mechanism for creating user interfaces in Windows applications. They allow you to encapsulate specific functionality with regard to its graphical appearance on the screen, its response to user input, and its method of notifying another window when an important input event has occurred.

Creating Child Window Controls

There are two main ways to create child window controls:

Manually: You can create child window controls manually by defining a window class and registering it with Windows using RegisterClass. You then create the child window based on that class using CreateWindow.



Using predefined controls: Windows provides a set of predefined child window controls that you can use without having to define your own window class. These controls include buttons, check boxes, edit boxes, list boxes, combo boxes, text strings, and scroll bars. To create a predefined child window control, you simply use the name of the control as the window class parameter in CreateWindow.



Communication between Child Window Controls and Parent Windows

Child window controls communicate with their parent windows using messages. The child window control sends messages to the parent window to notify it of important events, such as a button being clicked or a value being changed in an edit box. The parent window sends messages to the child window control to set its properties, such as its text or its enabled state.



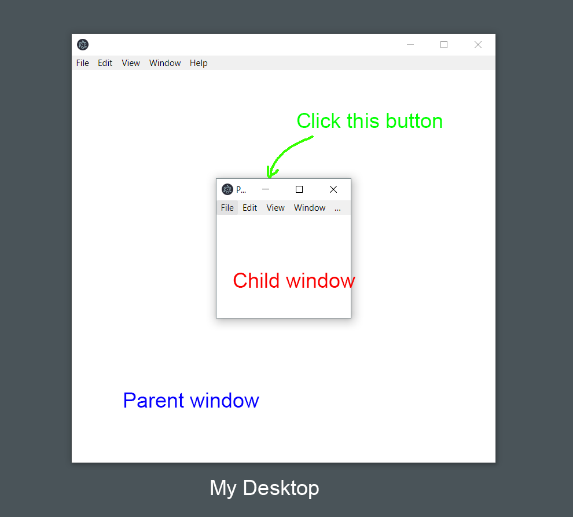
Child Window Controls in Dialog Boxes

Child window controls are used extensively in dialog boxes. The dialog box manager handles the placement and sizing of the child window controls, and it also provides a layer of insulation between your program and the controls themselves. This makes it easier to create dialog boxes without having to worry about the low-level details of child window controls.



Child Window Controls on Normal Windows

You can also use child window controls on the surface of a normal window's client area. However, this involves more work than using child window controls in dialog boxes, because you have to handle the placement and sizing of the child window controls yourself. You also have to handle the input focus, which can be a challenge.



Common Controls

Windows provides a set of specialized child window controls that are collectively known as "common controls." These controls are more complex than the simple standard controls, and they provide additional functionality, such as the ability to display images and to handle drag-and-drop operations.



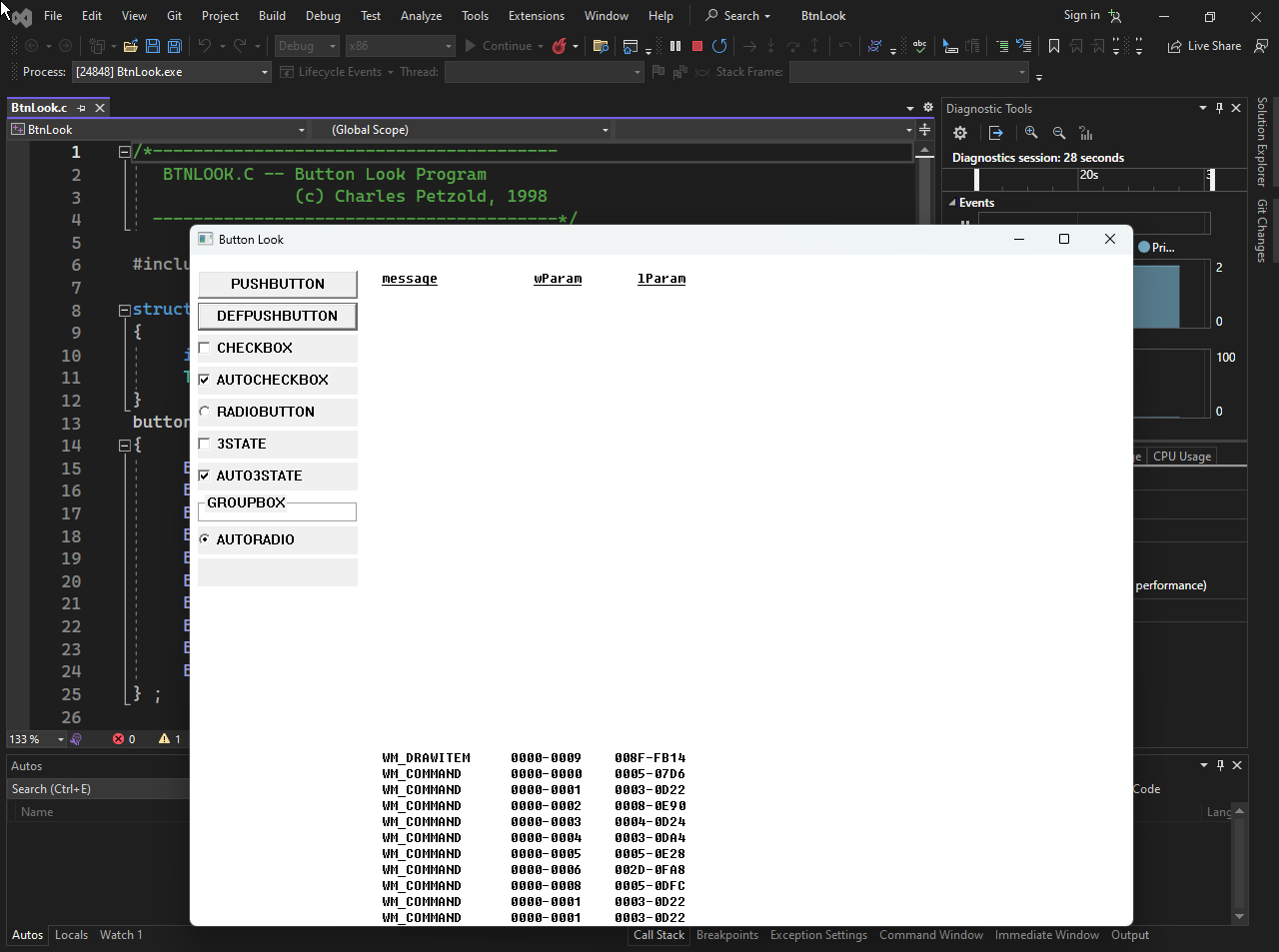
Additional Notes

The Windows programming documentation discusses child window controls in two places:

* Simple standard controls: These controls are described in /Platform SDK/User Interface Services/Controls.
* Common controls: These controls are described in /Platform SDK/User Interface Services/Shell and Common Controls/Common Controls.

I won't be discussing the common controls in this chapter, but they'll appear in various programs throughout the rest of the book.

*BtnLook program in chapter 9…*



The video illustration…



BTNLOOK Program Overview

The BTNLOOK program creates 10 child window button controls, one for each of the 10 standard styles of buttons. It displays the wParam and lParam parameters of the WM\_COMMAND messages sent by the buttons to the parent window procedure. The button with the style BS\_OWNERDRAW is displayed with a background shading because this is a style of button that the program is responsible for drawing.

Key Functionalities

Creates 10 child window button controls using the CreateWindow function.

* Handles WM\_CREATE, WM\_SIZE, WM\_PAINT, WM\_DRAWITEM, WM\_COMMAND, and WM\_DESTROY messages.
* Displays the wParam and lParam parameters of the WM\_COMMAND messages sent by the buttons.
* Handles owner-draw buttons, which are buttons that the program is responsible for drawing.
* The WndProc function handles all of the window messages for the main window.
* The CreateWindow function is used to create the child window button controls.
* The WM\_CREATE message handler creates the child window button controls and sets their initial positions.
* The WM\_SIZE message handler updates the positions of the child window button controls when the window is resized.
* The WM\_PAINT message handler draws the background of the window and the text labels for the buttons.
* The WM\_DRAWITEM message handler is sent to the owner-draw button, and it is responsible for drawing the button.
* The WM\_COMMAND message handler is sent to the parent window procedure whenever a button is clicked.
* The WM\_DESTROY message handler cleans up the resources used by the program and posts a WM\_QUIT message to the message queue.

Additional Notes:

* The program uses the GetDialogBaseUnits function to get the character size for the system font.
* The program uses the ScrollWindow function to scroll the contents of the client area when the buttons are resized.
* The program uses the InvalidateRect function to invalidate the client area when the buttons are clicked.

Child Windows

Child windows are windows that are created within the client area of another window, called the parent window. Child windows are typically created to provide additional functionality or content to the parent window. For example, a button on a dialog box is a child window of the dialog box.

Creating Child Windows

Child windows are created using the CreateWindow function. The CreateWindow function takes a number of parameters, including the following:

* Class name: The name of the window class. The window class defines the default appearance and behavior of the window.
* Window text: The text that will be displayed in the window's title bar.
* Window style: A set of flags that determine the appearance and behavior of the window.
* x position: The x-coordinate of the upper-left corner of the window's client area relative to the upper-left corner of the parent window's client area.
* y position: The y-coordinate of the upper-left corner of the window's client area relative to the upper-left corner of the parent window's client area.
* Width: The width of the window's client area.
* Height: The height of the window's client area.
* Parent window: The handle to the parent window.
* Child window ID: An ID that identifies the child window.
* Instance handle: The instance handle of the application.
* Extra parameters: Additional parameters that can be specified for certain types of windows.

Creating Buttons

Buttons are a common type of child window. To create a button, you can use the CreateWindow function and specify the following parameters:

* Class name: TEXT("button")
* Window text: The text that will be displayed on the button.
* Window style: WS\_CHILD | WS\_VISIBLE | BS\_DEFPUSHBUTTON
* x position: The x-coordinate of the upper-left corner of the button relative to the upper-left corner of the parent window's client area.
* y position: The y-coordinate of the upper-left corner of the button relative to the upper-left corner of the parent window's client area.
* Width: The width of the button.
* Height: The height of the button.
* Parent window: The handle to the parent window.
* Child window ID: The ID of the button.
* Instance handle: The instance handle of the application.
* Extra parameters: NULL

Processing Child Window Messages

Child windows send messages to their parent window to communicate with it. The parent window is responsible for processing these messages. To process child window messages, the parent window's window procedure must handle the WM\_COMMAND message.

Destroying Child Windows

Child windows are destroyed when their parent window is destroyed. You can also destroy a child window explicitly using the DestroyWindow function.

Additional Notes

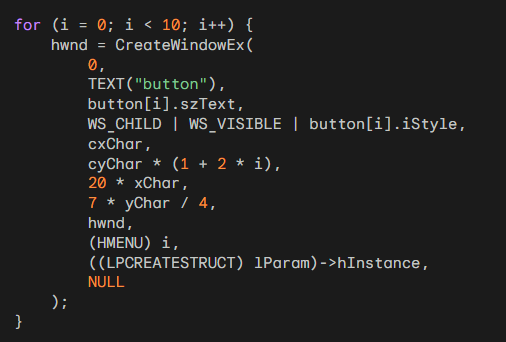
* Child windows can be nested. This means that a child window can be the parent of other child windows.
* Child windows can be modal or non-modal. Modal child windows prevent the user from interacting with other windows until the modal child window is closed.
* Child windows can be repainted by the parent window or by the system.

The code that creates the child windows is located in the WndProc function.

Specifically, the code is located in the WM\_CREATE message handling block. The code creates 10 child windows, one for each of the 10 button styles. The code uses the CreateWindowEx function to create the child windows. The CreateWindowEx function takes the following parameters:

* hParent: The handle to the parent window.
* hInstance: The instance handle of the application.
* lpClassName: The name of the window class.
* lpWindowName: The text that will be displayed in the window's title bar.
* dwStyle: A set of flags that determine the appearance and behavior of the window.
* x: The x-coordinate of the upper-left corner of the window's client area relative to the upper-left corner of the parent window's client area.
* y: The y-coordinate of the upper-left corner of the window's client area relative to the upper-left corner of the parent window's client area.
* cx: The width of the window's client area.
* cy: The height of the window's client area.
* hMenu: The handle to the window's menu.
* lpCreateStruct: A pointer to a CREATESTRUCT structure.
* lpvParam: An optional pointer to extra parameters.

The code that creates the child windows is as follows:



This code creates a child window for each of the 10 button styles. The code uses the following parameters:

* lpClassName: The name of the window class is TEXT("button").
* lpWindowName: The text that will be displayed in the window's title bar is:

button[i].szText.

* dwStyle: The window style is WS\_CHILD | WS\_VISIBLE | button[i].iStyle.
* x: The x-coordinate of the upper-left corner of the window's client area is cxChar.
* y: The y-coordinate of the upper-left corner of the window's client area which is:

cyChar \* (1 + 2 \* i).

* cx: The width of the window's client area is 20 \* xChar.
* cy: The height of the window's client area is 7 \* yChar / 4.
* hParent: The handle to the parent window is hwnd.
* hMenu: The child window ID is (HMENU) i.
* lpCreateStruct: The instance handle of the application is:

((LPCREATESTRUCT) lParam)->hInstance.

The code creates the child windows in a for loop. The loop iterates over the 10 button styles. For each button style, the code creates a child window and then increments the i counter.

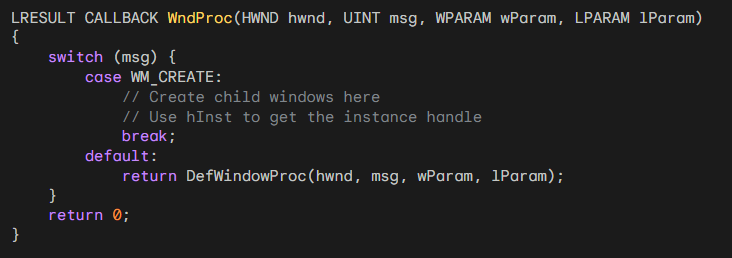
Here is an explanation of how to get the instance handle for a window procedure using a global variable named hInst. Create a global variable:



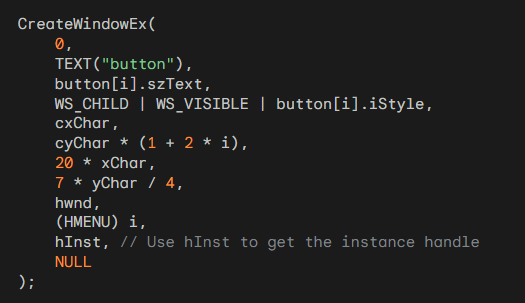
Set the global variable in WinMain:



Use the global variable in the window procedure:



Use the instance handle to create child windows:



Get the instance handle using GetWindowLong:



This code will get the instance handle for the window procedure and store it in the global variable hInst. The instance handle can then be used to create child windows and perform other tasks that require the instance handle.

When a button is clicked

When a button is clicked, the child window control sends a WM\_COMMAND message to its parent window. The WM\_COMMAND message is a notification message that is sent by a control window to its parent window to indicate that the control has been activated.

The WM\_COMMAND message has three parameters:

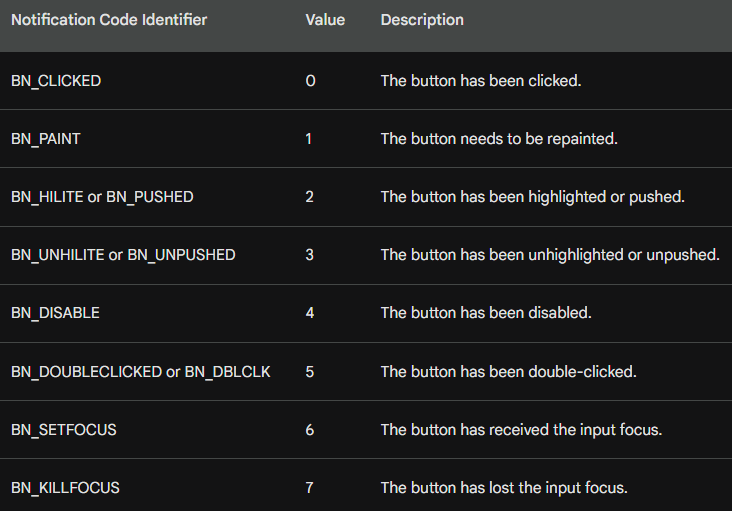
* wParam: The low-order word of wParam contains the child window ID. The high-order word of wParam contains the notification code.
* lParam: The lParam parameter contains the handle of the child window.

Child window ID

The child window ID is the value that is passed to the CreateWindow function when the child window is created. In BTNLOOK, the child window IDs are 0 through 9 for the 10 buttons that are displayed in the client area.

Notification code

The notification code indicates in more detail what the WM\_COMMAND message means. The possible values of button notification codes are defined in the Windows header files. The following table shows the notification codes that are used by BTNLOOK:



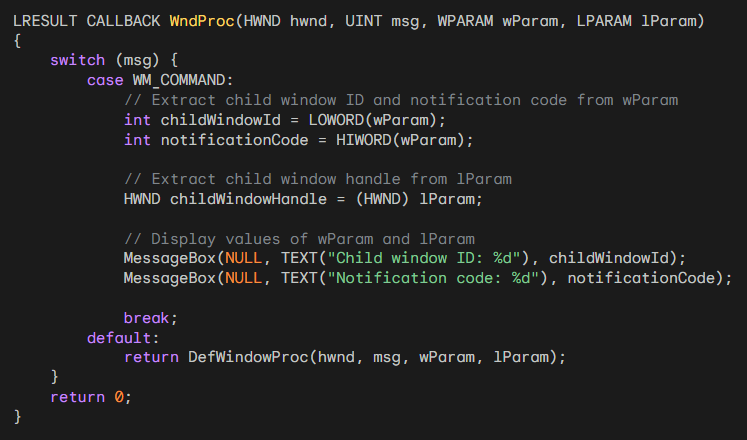
Handling WM\_COMMAND messages

The parent window of the child window control is responsible for handling WM\_COMMAND messages. BTNLOOK handles WM\_COMMAND messages by trapping the message in the WndProc function. The WndProc function then extracts the child window ID and notification code from the wParam parameter and the child window handle from the lParam parameter. The WndProc function then uses these values to display the values of wParam and lParam.

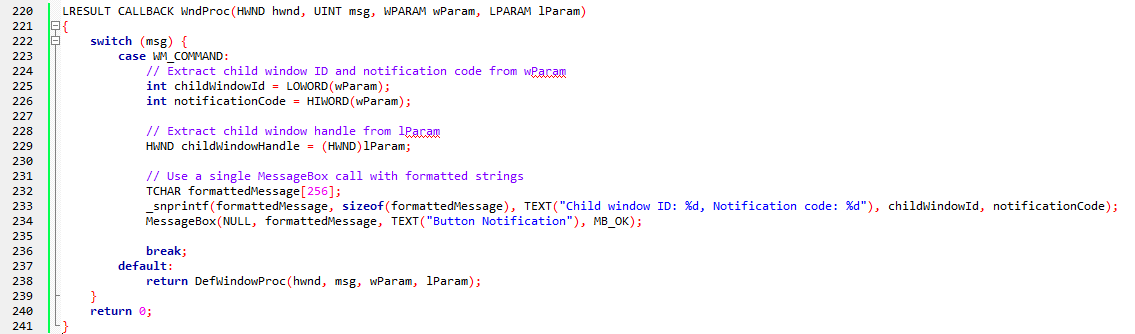
Input focus

When you click a button with the mouse, the button receives the input focus. This means that all keyboard input is now sent to the child window button control rather than to the main window. However, when the button control has the input focus, it ignores all keystrokes except the Spacebar, which now has the same effect as a mouse click.

The following code shows how BTNLOOK handles WM\_COMMAND messages:



Let’s correct this code…



Zoom to see the code. It was long, I had to put it that way.

This corrected code:

* Uses a single MessageBox call with formatted strings to display both the child window ID and notification code in a single message box. This improves readability and user experience.
* Replaces the MessageBox calls with a single call to avoid halting the program flow multiple times. This maintains the responsiveness of the program.
* Uses \_snprintf to format the message string into a buffer before passing it to MessageBox. This ensures that the formatted message fits within the buffer size.
* Uses TEXT macros for string literals to ensure compatibility with Unicode.