Child Window Controls: List Boxes, Combo Boxes, Scroll Bars, Edit Controls

Custom Child Windows

List Box Controls

- Lots of styles: see on-line help on LBS_
  - LBS_STANDARD very common
    - can send messages to parent
- Program communicates with list box by sending it messages; some common button messages:
  - LB_RESETCONTENTS, LB_ADDSTRING, LB_GETCURSEL, LB_GETTEXT, LB_DELETESTRING
- Some List Box Notification codes:
  - LBN_SELCHANGE, LBN_DBLCLK
- Combo boxes much like list boxes (CBS_, CB_, CBN_)
- Program examples: listbox, combo

Messages from Most Controls

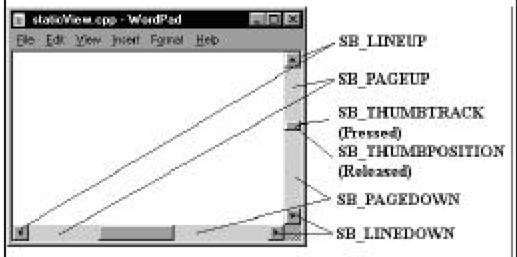
- Most work as follows:
  - User interacts with the control
  - WM_COMMAND message sent to parent window
  - LOWORD(wParam) = Control ID
  - lParam = control's window handle
  - HIWORD(wParam) = notification code
    - identifies what the user action was
- Scroll Bars are a bit different

Scroll Bar Controls

- User interacts with a scroll bar
  - WM_HSCROLL or WM_VSCROLL message
  - Not WM_COMMAND as for other controls
    - lParam =scroll bar window handle (for stand-alone)
    - lParam =0 (for attached scroll bar)
    - LOWORD(wParam)=notification code: user action
      - SB_LINEUP (up/left arrow pressed)
      - SB_PAGEUP (scroll area above/left of "thumb")
      - SB_LINEDOWN (down/left arrow pressed)
      - SB_PAGEDOWN (scroll area beneath/right of "thumb")
      - SB_THUMBTRACK (scroll "thumb" pressed)
      - SB_THUMBPOSITION (scroll "thumb" released)
        - For either, HIWORD(wParam)=current thumb position

- Lots of Scroll bar styles when creating it
  - See online help on SBS_
  - Default alignment for attached scroll bar: right side and bottom of window
- **Some Useful Scrollbar Functions:**
  - ***GetScrollPos()***--retrieve current position of thumb
    GetScrollRange ()--Retrieve min/max value range
    SetScollPos() --Set position of thumb
    SetScrollRange()--Set min/max value range
    ShowScrollBar ()--Display scroll bar
    - 1st params: hWnd or hScrollBar
    - 2nd param: SB_CTL (standalone) or SB_VERT/SB_HORZ (attached scroll bar)
    - Others: position, range (2 values), etc…, visibility flag

Scroll Bar Notification Codes



Scroll Bar Notification codes -- LOWORD (lParam)

The SCROLL1 Example

- Win32 API Application
- Stand-alone scrollbar allows user to enter an integer value between 0 and 50
- Current value is continually displayed in a static control
- Message box shows current value when user chooses menu item "Get Value"
- See Scroll1 code on Example Programs web page

The SCROLL2 Example

- Win32 API Application
- Scroll Bar Attached to a Window
- Creates a window with a vertical scroll bar
- Puts 3 lines of text in client area
- User can scroll through the client area using scroll bar
  - Opposite direction from "normal" scrolling
- See Scroll2 code on Example Programs web page

CScrollbar Class for Standalones

- In Create() member function, include SB_HORZ or SB_VERT style
- Make calls to member functions:
  - ,***SetScollPos()***, ***SetScrollRange (), etc.***
- Include ON_WM_HSCROLL or ON_WM_VSCROLL message mapping macros
- Override Handler, e.g.:

  afx_msg void OnHScroll (UINT nCode, UINT nPos, CScrollBar* pScrollBar);
  - nCode= SB_*** notification code (user action)
  - nPos=latest thumb position for drags/releases
  - pointer to the scroll bar

Attached Vertical Scroll Bar in Doc/View MFC Apps

- Override View class's OnCreate(...) member function to set range and position of vertical scroll bar
- Use Class Wizard to add: ON_WM_VSCROLL() message mapping macro and OnVScroll(...) handler function in View class
  - Add switch/case statements to handle SB_codes of interest…in OnVScroll() handler function
- See Scroll2_mfc Example Program

EDIT CONTROLS

- **For viewing and editing text**
- Current location kept track of with a "carat"
  - A small vertical line
- Backspace, Delete, arrow keys, highlighting work as expected
- Scrolling possible (use WS_HSCROLL, WS_VSCROLL styles
- No ability to format text with different fonts, sizes, character styles, etc.
  - Use Rich Edit Control for this

Edit Control Styles

- Some common styles
  - ES_LEFT, ES_CENTER, ES_RIGHT, ES_MULTILINE, ES_AUTOVSCROLL, ES_PASSWORD
    - See Online Help on "Edit Styles"

Edit Control Text

- Text in an edit control stored as one long character string
- Carriage return <CR> is stored as ASCII code (0x0D,0x0A)
- <CR> inserted automatically if a line doesn't fit and wraps
- NULL character inserted only at end of last line of text

Edit Control Messages

- User interacts with edit control,
  - WM_CONTROL message to parent
  - LOWORD(wParam) = Control ID
  - lParam = control's window handle
  - HIWORD(wParam) = EN_** notification code
    - identifies what the user action was
    - e.g., EN_CHANGE
    - See Online Help EN_***
- MFC: Add to message map and add handler:
    - ON_Notification**(** *id, memberFtn* **)**
    - afx_msg void *memberFtn* ***( );***

Sending Messages to an Edit Box

- As with other controls use SendMessage()
- Some important messages
  - EM_GETLINECOUNT(multiline edit boxes)
    - Returns number of lines in the control
  - EM_GETLINE: Copy a line to a buffer
  - EM_LINEINDEX: Get a line's character index
    - Number of characters from the beginning of edit control to start of specified line
  - EM_LINELENGTH to get length of line
- See Edit1 example program

MFC's CEdit Class

- Some important member or inherited functions
  - SetWindowText(LPSTR)
    - Place text in the control
    - Replaces current contents
    - Could be a CString
  - GetWindowText(LPSTR)
    - Returns all the lines in the control
    - Could be a Cstring
  - Lots of others, see Online Help on CEdit

Child and Popup Windows

- Child Window Controls are predefined window controls (buttons, static text, etc.)
  - These are examples of **child windows**
- OK if controls have exact features required
- But sometimes we need custom child windows
  - Where we can have a WndProc() that does exactly what we want it to

Child Window

- Most common type of custom window
- Always attached to parent window
  - Always on top of parent
  - Parent minimized ➔ child disappears
  - Reappears when parent restored
  - Parent destroyed ➔ child also destroyed
- Used to deal with a specific task
  - e.g., getting user input
- Each has its own message-processing function

Popup window

- Same general properties as child window, but:
- Not physically attached to parent
- Can be positioned anywhere on screen
- Handy if the user needs to move things around on client area

Creating and Using a Child Window

- 1. Register a new window class for child using ***RegisterClass()***
  - Could be done in ***WinMain()*** or when needed in ***WndProc()***
- 2. Create child window using ***CreateWindow()***
  - Should have WS_CHILD style
- 3. Write separate message-processing function for child window

Sending Messages to a Child Window

- Use ***SendMessage()*** and specify:
  - Child window's handle
    - Obtained when the child window was created
  - Message ID & parameters

WM_USER Messages

- Defined in Windows.h as a number not used by predefined messages
- All higher numbers also unused by Windows
- Can use WM_USER + # for any type of activity
- Example--could have a header file containing:

```
#define WM_MYKILLCHILD   WM_USER
   // tell child window to vanish
#define WM_MYMAXCHILD    WM_USER+1
   // tell child window to maximize
```

  Use in child's WndProc() function’s switch/case
- Child windows can send messages to parent or to other child windows

CHILD EXAMPLE PROGRAM

- User clicks "Create" menu item➔
  - Child window appears with "Destroy Me" button and some text
- User clicks "Send Message" menu➔
  - Caption on child window changes
- User clicks "Destroy Me" button in child window➔
  - Child window disappears
- Both parent and child window have a line of text displayed in client areas

Details of CHILD Application

- **1.Register Child Window Class with *RegisterClass()***
  - Message processing function: *ChildProc()*
  - Will receive messages from any windows based on this class
  - Class Icon: IDI_APPLICATION icon
  - Cursor shape: Load standard IDC_CROSS cursor
  - Background: LTGRAY_BRUSH background brush
  - Menu: None to be used here

- 2. Create Child Window using *CreateWindow()*
- 3. Menu item response
  - User clicks "Create" menu item (WM_COMMAND, IDM_CREATE)➔
    - Program's *WndProc()* executes:

```
if(!hChild)
    hChild = CreateWindow ("ChildClass",
     "Child Window", WS_CHILD |
     WS_THICKFRAME | WS_MINIMIZEBOX |
     WS_MAXIMIZEBOX | WS_CAPTION |
     WS_SYSMENU, 10, 30, 200, 150, hWnd,
     NULL, hInstance, NULL);
```

  - Logic allows only one child window at a time

Sending Messages

- In Main Window’s *WndProc()*
  - User clicks "Send Message" menu item➔
  - *WndProc()* uses ***SendMessage()*** to send a WM_USER msg to child window

- *In Child’s ChildProc()*
  - *ChildProc()*'s response to WM_USER from parent:
    - Uses ***SetWindowText()*** to set its caption bar
  - Response to creation:
    - ***CreateWindow()*** to create a "Destroy Me" pushbutton
    - 3-deep nesting of windows: Parent (main window), Child Window, Button Control
  - Response to expose event:
    - Output a line of text to child window client area
  - Response to user clicking the pushbutton:
    - Use ***GetParent(hChild)*** to get the parent's window handle
    - Destroys itself with a call to ***DestroyWindow(hChild)***
    - Send USER+1 message to parent

- Main Window’s *WndProc()'s* response to this (WM_USER+1):
  - Set hChild to NULL so another child can be created
  - *WndProc()* also responds to expose events by outputting a line of text to main window's client area
  - So text in both windows is visible whenever either is exposed

POPUP WINDOWS

- Not restricted to the parent window's client area
- Can appear anywhere on screen
- Handy for small utility programs
  - e.g., Window that shows current cursor position in a painting program
- Ideal for applications with multiple independent sections, e.g.:
  - Communications program with simultaneous terminal sessions in different popup windows
- Create with ***CreateWindow()***
  - WS_POPUP style (mutually exclusive with WS_CHILD)
  - Coordinates are screen coordinates