Latest updated by: HanhTT1

Agenda

- Get Started
- The Button Class
- The Static Class
- The Scroll Bar Class
- The Edit Class
- The List Box Class

Get Started

- Child window controls are windows and have window procedures
- Child window can get parent window's handle by call function **GetParent()** :

  hWndParent = GetParent(hWnd); // hWnd is handle of child window

- Child window can send messages to parent window by call function **SendMessage()** :

  SendMessage (hWndParent, message, wParam, lParam);

- Child window processes mouse, keyboard messages and notifies the parent window when the child window's state has changed

Get Started

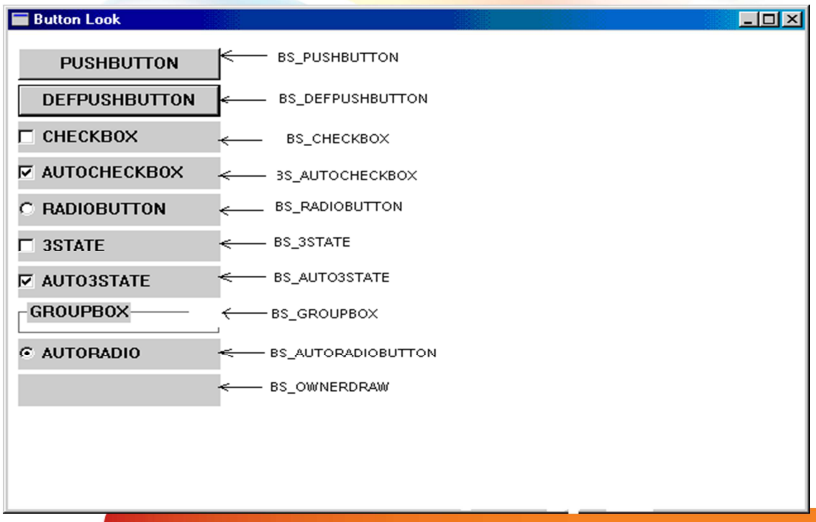
- Windows has predefined window classes and window procedures, for example : buttons, check boxes, edit boxes, list boxes, combo boxes, scroll bars, text strings
- Child window controls are used most often in dialog boxes. However, we could use predefined child window controls on a surface of a normal window's client area
- Create each child window with a **CreateWindow()** call and adjust the position and size of the child window with calls to **MoveWindow()**
- The parent window procedure sends messages to child window controls, child window controls send messages back to the parent window procedure

Get Started

- When use one of the predefined controls, do not register a window class for the child window because the class is already existed within Windows and has a predefined name
- Windows contains the window procedures that process messages to the child windows based on these predefined classes
- When use child window controls directly on the surface of a window, these child window controls have no built-in facility to move the input focus from one control to another using the Tab or cursor movement keys
- A child window control can obtain the input focus, but once it does it won't freely relinquish the input focus back to the parent window

Basic Child Window Controls

Child window controls



Create an windows control

- Three are two main ways you can create a control
  - ✓ Create a control in design time: this is a simplest way
  - ✓ Create a control using code: this is more complex, we will study later

Create a control programmatically

- Create a control (Combo box) with calls to **CreateWindow()** function uses the following parameters :
  - Class name **_T("COMBOBOX")**
  - Window text **Combo's caption**
  - Window style **WS_CHILD | WS_VISIBLE | WS_TABSTOP…**
  - X Position Left corner of combo
  - Y Position Left corner of combo
  - Width Combo's width
  - Height Combo's heigth
  - Parent window **Handle of parent window**
  - Child window ID **ID of combo**
  - Instance handle **Instance handle of application**
  - Extra parameters NULL

Create a control programmatically - Exp

```
LRESULT CALLBACK DlgProc(HWND hWndDlg, UINT Msg, WPARAM wParam, LPARAM lParam) {
   switch(Msg) {
      case WM_INITDIALOG:
            hWndComboBox = CreateWindow("COMBOBOX", NULL, WS_CHILD | WS_VISIBLE |
                                                                  WS_TABSTOP, 60, 62, 136, 60, hWndDlg, NULL,
      hInst, NULL);
            if (!hWndComboBox ) {
               MessageBox(hWndDlg, "Could not create the combo box", "Failed Control Creation", MB_OK);
                                    return FALSE;
            }
            return TRUE;
      case WM_COMMAND:
            switch (wParam) {
               case IDCANCEL:
                  EndDialog(hWndDlg, 0);
                  return TRUE;
            }
            break;
      }
      return FALSE;
}
```

Working with child window controls

- After create controls with calls to **CreateWindow()**, needn't do anything more with these controls. The control window procedure within Windows maintains the control for us and handles all repainting jobs
- At the program's termination, Windows destroys these controls when the parent window is destroyed

Working with windows controls (2)

- When you click a control with the mouse, the child window control sends a **WM_COMMAND** message to its parent window
- This message is processed in window procedure of parent window
- The meaning of **wParam** and **lParam** parameters of **WM_COMMAND** message :

  LOWORD (wParam) ID of child window

  HIWORD (wParam) Notification code

  lParam Child window handle

- Notification code indicates in more detail what the message means

Notification Code

Child window Notificatin Code Identifier	Value
BN_CLICKED	0
BN_PAINT	1
BN_HILITE or **BN_PUSHED**	2
BN_UNHILITE or **BN_UNPUSHED**	3
BN_DISABLE	4
BN_DOUBLECLICKED or **BN_DBLCLK**	5
BN_SETFOCUS	6
BN_KILLFOCUS	7

- The notification codes 1 through 4 are for an obsolete control style called BS_USERBUTTON and it's been replaced with BS_OWNERDRAW and a different notification mechanism
- The notification codes 6 and 7 are sent only if the control style includes the flag BS_NOTIFY
- The notification code 5 is sent only for BS_RADIOBUTTON, BS_AUTORADIOBUTTON, and BS_OWNERDRAW controls, or for other controls if the control style includes BS_NOTIFY

Child window messages

- A window procedure can send messages to the child window control
- In addition, Windows has defined eight control-specific messages in WINUSER.H

Control Message	***Value***
BM_GETCHECK	0x00F0
BM_SETCHECK	0x00F1
BM_GETSTATE	0x00F2
BM_SETSTATE	0x00F3
BM_SETSTYLE	0x00F4
BM_CLICK	0x00F5
BM_GETIMAGE	0x00F6
BM_SETIMAGE	0x00F7

Push Buttons control

- When click on push button, button sends a **WM_COMMAND** message to parent window with notification code is **BN_CLICKED**
- Causes the button to be depressed :

  SendMessage (hWndButton, BM_SETSTATE, 1, 0);

- Causes the button to return to normal :

  SendMessage (hWndButton, BM_SETSTATE, 0, 0);

Check boxes control (1)

- The two most common styles for a check box are **BS_CHECKBOX** and **BS_AUTOCHECKBOX**
- With **BS_CHECKBOX**, must set the check mark by sending to control a **BM_SETCHECK** message with **wParam** parameter is set to 1 to create a check mark and to 0 to remove it

```
SendMessage (hWndButton, BM_SETCHECK, 1, 0);
SendMessage (hWndButton, BM_SETCHECK, 0, 0);
```

- Obtain the current check state of the check box by sending to control the **BM_GETCHECK** message

```
SendMessage (hWndButton, BM_GETCHECK, 0, 0);
```

Check boxes control (2)

- With **BS_AUTOCHECKBOX**, the check box control itself toggles the check mark on and off. Window procedure can ignore **WM_COMMAND** message
- Get current state of the check box, send to control a **BM_GETCHECK** message

  iCheck = (int) SendMessage (hWndButton, **BM_GETCHECK**, 0, 0);

  iCheck = TRUE (!= 0) : button is checked
  iCheck = FALSE (== 0) : button is not checked
- With **BS_3STATE**, **BS_AUTO3STATE** styles, the control can display a third state, occurs when sends to control a **WM_SETCHECK** :

  SendMessage (hWndButton, **BM_SETCHECK**, **2**, 0);

Radio Buttons control

- The radio button has the window style **BS_RADIOBUTTON** or **BS_AUTORADIOBUTTON**, but the latter is used only in a dialog boxes
- When receives a **WM_COMMAND** message from a radio button, should display its check by sending a **BM_SETCHECK** with **wParam** is set to 1

  SendMessage (hWndButton, **BM_SETCHECK**, **1**, 0);

- With all radio buttons in a group, could turn off the checks by sending them **BM_SETCHECK** with wParam is set to 0

  SendMessage (hWndButton, **BM_SETCHECK**, **0**, 0);

Group boxes control

- The group box has BS_GROUPBOX style
- The group box neither processes mouse or keyboard input nor sends **WM_COMMAND** messages to its parent
- Group boxes are often used to enclose other button controls

Changing the control's Text

- Change the text in a control (or in any other window) by calling **SetWindowText** :

  SetWindowText (HWND hWnd, LPCTSTR lpSzString);

- Obtain the current text of a window :

  iLength = GetWindowText (HWND hWnd, LPCTSTR

  lpSzString, int nMaxLength);

- Prepare your program for a particular text length by first calling :

  iLength = GetWindowTextLength (HWND hWnd);

Visible and Enabled controls (1)

- If you do not include **WS_VISIBLE** in the window class of child window control, child window will not be displayed until you make call to **ShowWindow** function :

  ShowWindow (hWndChild, SW_SHOWNORMAL);

- If you include WS_VISIBLE in the window class of child window control, then no need to call ShowWindow. However, you could hide the child window by this call to ShowWindow function :

  ShowWindow (hWndChild, SW_HIDE);

- Determine if the child window is visible or not by call to :

  IsWindowVisible (hWndChild);

Visible and Enabled controls (2)

- You could enable or disable a child window, by default, a window is enabled. You could disable it by a call to :

  EnableWindow (hWndChild, FALSE);

- Re-enable a child window by calling :

  EnableWindow (hWndChild, TRUE);

- Determine wheter a child window is enabled or not by calling :

  IsWindowEnabled (hWndChild);

Controls and Input Focus

- When the child window control gets the input focus, the parent window loses it. However, the child window control responds only to the Space bar
- When Windows switches the input focus from one window (such as parent) to another (such as child window control), it first sends **WM_KILLFOCUS** message to the window losing the input focus. The **wParam** is handle of the window that receiving input focus
- Windows then sends a **WM_SETFOCUS** message to the window receiving the input focus with **wParam** is handle of window that losing the input focus

Controls and Input Focus (2)

- A parent window could prevent a child window receive input focus by processing **WM_KILLFOCUS** message :

```
case WM_KILLFOCUS :

   for (i = 0 ; i < NUM ; i++)
      if (hwndChild [i] == (HWND) wParam) {
                  SetFocus (hwnd) ;
                  break ;
      }
   return 0 ;
```

The Static control

- The static child window controls are rectangles that contain text string on a window (often known as “**label**” control)
- Create static child window controls by using “**static**” as the window class in the **CreateWindow** function
- When moving or clicking the mouse over a static child window, child window traps a **WM_NCHITTEST** message and returns **HTTRANSPARENT** value to Windows. This make Windows sending the same **WM_NCHITTEST** message to parent window. The parent window usually passes this message to **DefWindowProc**
- Some static class styles : **SS_BLACKRECT**, **SS_GRAYRECT**, **SS_WHITERECT**, **SS_BLACKFRAME**, **SS_GRAYFRAME**, **SS_WHITEFRAME**,…

The Scroll Bar control

- Create child window scroll bar controls by using the predefined window class “**scrollbar**” and one of the two scroll bar styles **SBS_VERT** and **SBS_HORZ**
- The scroll bar controls do not send **WM_COMMAND** messages to the parent window. Instead, they send **WM_VSCROLL** or **WM_HSCROLL** messages. The **lParam** parameter is handle of child window scroll bar
- Set the range and position of a scroll bar control :

  SetScrollRange (**hWndScroll**, **SB_CTL**, iMin, iMax, bRedraw);
  SetScrollPos (**hWndScroll**, **SB_CTL**, iPos, bRedraw);
  SetScrollInfo (**hWndScroll**, **SB_CTL**, &si, bRedraw);

The Edit Class

- The edit child window controls are rectangles that contain editable text.
- Create edit child window control by using the class name “**edit**” in **CreateWindow** call
- Some edit window control styles : **ES_LEFT**, **ES_RIGHT**, **ES_CENTER, ES_MULTILINE, ES_AUTOHSCROLL, ES_AUTOVSCROLL, …**

The Edit Control Notification

- The edit controls send **WM_COMMAND** messages to parent window procedure
- The meaning of **wParam** and **lParam** parameters are the same as button controls :

LOWORD (wParam)	Child window ID
HIWORD (wParam)	Notification code
lParam	Child window handle

- List of notification codes :

EN_SETFOCUS	Edit control has gained the input focus
EN_KILLFOCUS	Edit control has lost the input focus
EN_CHANGE	Edit control's contents will change
EN_UPDATE	Edit control's contents have changed
EN_ERRSPACE	Edit control has run out of space
EN_MAXTEXT	Edit control has run out of space on insertion
EN_HSCROLL	Edit control's horizontal scroll bar has been clicked
EN_VSCROLL	Edit control's vertical scroll bar has been clicked

Using the Edit Controls

- Insert text into an edit control by using **SetWindowText** function
- Get text out of an edit control by using **GetWindowTextLength** and **GetWindowText** functions

Messages to an Edit Control

- The messages let you cut, copy, delete the current selection in edit control :

```
SendMessage (hWndEdit, WM_CUT, 0, 0);
SendMessage (hWndEdit, WM_COPY, 0, 0);
SendMessage (hWndEdit, WM_CLEAR, 0, 0);
```

- Insert clipboard text into edit control at the cursor position :

```
SendMessage (hWndEdit, WM_PASTE, 0, 0);
```

- Obtain the starting and ending positions of the current selection :

```
SendMessage (hWndEdit, EM_GETSEL, (WPARAM) & iStart,
(LPARAM) & iEnd);
```

Messages to an Edit Control

- Select text :

```
SendMessage (hWndEdit, EM_SETSEL, iStart, iEnd);
```

- Replace the current selection with other text :

```
SendMessage (hWndEdit, EM_REPLACESEL, 0, (LPARAM)
                                                szString);
```

- For particular line, obtain an offset from the beginning of the edit buffer text :

```
iOffSet = SendMessage (hWndEdit, EM_LINEINDEX, iLine, 0);
```

- For a particular line, obtain line's length :

```
iLength = SendMessage (hWndEdit, EM_LINELENGTH, iLine, 0);
```

- Copy the line itself into a buffer using :

```
iLength = SendMessage (hWndEdit, EM_GETLINE, iLine, (LPARAM)
                                                szBuffer);
```

- Get the number of lines with multi-line edit control :

```
SendMessage (hWndEdit, EM_GETLINECOUNT, 0, 0);
```

The List Box control

- A list box is a collection of text strings displayed as a scrollable columnar list within a rectangle
- A program can add or remove strings in the list by sending messages to the list box window procedure
- The list box control sends **WM_COMMAND** message to its parent window procedure when an item in the list is selected. The parent window can then determine which item has been selected
- A list box can be either single selection or multiple selection

The List Box control

- Create a list box child window control with **CreateWindow** using "**listbox**" as the window class and **WS_CHILD** as the window style
- For make list box child window controls send **WM_COMMAND** messages to its parent, include the list box style identifier **LBS_NOTIFY**
- Some list box window control styles : **LBS_SORT, LBS_MULTIPLESEL, LBS_NOREDRAW, WS_SIZEBOX, WS_CAPTION, …**

The List Box control

- Putting strings into list box :

```
SendMessage (hWndList, LB_ADDSTRING, 0,
                                    (LPARAM) szString);
SendMessage (hWndList, LB_INSERTSTRING,
                                    iIndex, (LPARAM) szString);
```

- Delete a string from a list box :

```
SendMessage (hWndList, LB_DELETESTRING, iIndex,
                                    0);
```

- Delete all contents of list box :

```
SendMessage (hWndList, LB_RESETCONTENT, 0, 0);
```

The List Box control

- Find out how many items in list box :

  iCount = SendMessage (hWndList, LB_GETCOUNT, 0, 0);

- Highlight a default selection :

  SendMessage (hWndList, LB_SETCURSEL, iIndex, 0);

- Determine the index of current selection :

  iIndex = SendMessage (hWndList, LB_GETCURSEL, 0, 0);

- Copy the item into the text buffer :

  iLength = SendMessage (hWndList, LB_GETTEXT, iIndex, (LPARAM) szBuffer);

- Determine the length of any strings in list box :

  iLength = SendMessage (hWndList, LB_GETTEXTLEN, iIndex, 0);

- Select one item base on its initial character :

  SendMessage (hWndList, LB_SELECTSTRING, iIndex, (LPARAM) szSearchString);

- With multiple selection list box, set the selection state of a particular item without affecting other items that might also be selected :

  SendMessage (hWndList, LB_SETSEL, wParam, iIndex);

The List Box control

- Receiving messages from list boxes : a list box control sends **WM_COMMAND** messages to its parent. The meaning of **wParam** and **lParam** parameters are same as edit and button controls :

LOWORD (wParam)	Child window ID
HIWORD (wParam)	Notification code
lParam	Child window handle

- List of notification codes and their values :

LBN_ERRSPACE	-2
LBN_SELCHANGE	1
LBN_DBLCLK	2
LBN_SELCANCEL	3
LBN_SETFOCUS	4
LBN_KILLFOCUS	5


Q&A