**Child Window Controls**

- Windows created by a parent window
- An app uses them in conjunction with parent
- Normally used for simple I/O tasks
- Let user choose commands, view status, view/edit text, etc.
- Properties, appearance, behavior determined by predefined class definitions
  - But behavior can be customized
  - Easy to set them up as common Windows objects
    - buttons, scroll bars, etc.

- Allow user to display/select info in standard ways
- Windows Environment does most of work in:
  - painting/updating a Control’s screen area
  - determining what user is doing
- Can do the “dirty work” for the main window
- Often used as input devices for parent window
- Are the “working components” of dialog boxes
- Windows OS contains each control’s WinProc
  - so messages to controls are processed in predefined way
- Parent window communicates with controls by sending/receiving messages

- Windows Environment automatically repaints a Control upon exposure
- Example: WordPad (“File”|“Open”)
  - Contains most of “classic” controls
  - There are 20 other predefined “Common Controls”
  - Most first appeared in Windows 95
  - Some came with Internet Explorer
  - Implemented in Comctl32.dll

---

**Six “Classic” Control Types**

- Go back to first versions of Windows
- Implemented in User.exe

<table>
<thead>
<tr>
<th>Type</th>
<th>Window Class</th>
<th>MFC Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static Text</td>
<td>“STATIC”</td>
<td>CS tatic</td>
</tr>
<tr>
<td>Button</td>
<td>“BUTTON”</td>
<td>CButton</td>
</tr>
<tr>
<td>Edit Control</td>
<td>“EDIT”</td>
<td>CEdit</td>
</tr>
<tr>
<td>List Box</td>
<td>“LISTBOX”</td>
<td>CListBox</td>
</tr>
<tr>
<td>Combo Box</td>
<td>“COMBOBOX”</td>
<td>CComboBox</td>
</tr>
<tr>
<td>Scroll Bar</td>
<td>“SCROLLBAR”</td>
<td>CScrollBar</td>
</tr>
</tbody>
</table>

- All are windows

---

**The Common Controls**

<table>
<thead>
<tr>
<th>TYPE</th>
<th>WINDOW CLASS</th>
<th>MFC CLASS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Animation</td>
<td>“SysAnimate32”</td>
<td>CAnimateCtrl</td>
</tr>
<tr>
<td>ComboBoxEx</td>
<td>“ComboBoxEx32”</td>
<td>CComboBoxEx</td>
</tr>
<tr>
<td>Date-Time</td>
<td>“SysDateTimePick32”</td>
<td>CDateTimeCtrl</td>
</tr>
<tr>
<td>Header</td>
<td>“SysHeader”</td>
<td>CHeaderCtrl</td>
</tr>
<tr>
<td>Hotkey</td>
<td>“msctls_hotkey32”</td>
<td>CHotKeyCtrl</td>
</tr>
<tr>
<td>Image List</td>
<td>N/A</td>
<td>CImageList</td>
</tr>
<tr>
<td>IP Address</td>
<td>“SysIPAddress32”</td>
<td>CIPAddressCtrl</td>
</tr>
<tr>
<td>List View</td>
<td>“SysListView32”</td>
<td>CListCtrl</td>
</tr>
<tr>
<td>Month Calendar</td>
<td>“SysMonthCal32”</td>
<td>CMonthCalCtrl</td>
</tr>
<tr>
<td>Progress</td>
<td>“msctls_progress32”</td>
<td>CProgressCtrl</td>
</tr>
<tr>
<td>Property Sheet</td>
<td>N/A</td>
<td>CPropertySheet</td>
</tr>
<tr>
<td>ReBar</td>
<td>“ReBarWindows32”</td>
<td>CReBarCtrl</td>
</tr>
<tr>
<td>TYPE</td>
<td>WINDOW CLASS</td>
<td>MFC CLASS</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>Slider</td>
<td>&quot;msctls_trackbar32&quot;</td>
<td>CSliderCtrl</td>
</tr>
<tr>
<td>Spin Button</td>
<td>&quot;msctls_updown32&quot;</td>
<td>CSpinButtonCtrl</td>
</tr>
<tr>
<td>Status Bar</td>
<td>&quot;msctls_statusbar32&quot;</td>
<td>CStatusBarCtrl</td>
</tr>
<tr>
<td>Tab</td>
<td>&quot;SysTabControl32&quot;</td>
<td>CTabCtrl</td>
</tr>
<tr>
<td>Toolbar</td>
<td>&quot;ToolbarWindow32&quot;</td>
<td>CToolBarCtrl</td>
</tr>
<tr>
<td>ToolTip</td>
<td>&quot;tooltips_class32&quot;</td>
<td>CToolTipCtrl</td>
</tr>
<tr>
<td>Tree View</td>
<td>&quot;SysTreeView32&quot;</td>
<td>CTreeCtrl</td>
</tr>
</tbody>
</table>

**Classic Window Controls**

- **Static**
  - Primarily to display text
  - Can also display icon images and rectangles
  - Automatically redrawn if exposed
  - Often used as labels for other controls

- **Button**
  - "Clicked" by user to indicate desired actions or choices made
  - Lots of different styles (e.g., pushbutton, check, radio, group)
  - Typically notify parent window when user chooses the button

- **List Box**
  - Contains lists of items that can be selected
  - Entire list is shown
  - User selects items
  - Selected item is highlighted

- **Combo Box**
  - Edit box combined with a list box
  - List box can be displayed at all times or pulled down
  - User selects item from list & item is copied to edit box
  - One type allows user to type into edit box
  - Another type doesn’t allow user to type in edit box

- **Scroll Bar**
  - Lets user choose direction/distance to move a “thumb”
  - Two types:
    - Control attached to edge of a parent window
    - Allows user to “scroll” the information in a parent window’s client area
  - Stand-alone child window control
  - Allows user to enter/change a value by moving scroll bar “thumb”

- **Edit**
  - To enter/view/edit/delete text
  - Single or multiline control
  - Lots of word processing capability
  - Also Clear/Copy/Cut/Paste/Undo capability

---

**Creating Controls--Win32 API**

- **CreateWindow()**
  - For any kind of window, including a control
  - Typically called in response to WM_CREATE

- **Parameters:**
  - 1. Predefined control class names:
    - "STATIC", "BUTTON", "EDIT", "LISTBOX", "COMBOBOX", "SCROLLBAR", others
  - 2. Name of the window
    - BUTTON, EDIT, STATIC classes:
      - text in center of control
    - COMBOBOX, LISTBOX, SCROLLBAR:
      - ignored (use "")
  - 3. Window style
    - WS_, SS_, BS_, ES_, LBS_, CBS_, SBS_ (see CreateWindow help)
    - Several styles can be combined with the bitwise or operator (∨)
    - All controls should include WS_CHILD style

- **Parameters 4-7:**
  - X,Y position (Relative to the upper left corner of parent window client area)
  - Width & Height

- **8. Handle to the parent window**
9. Handle to “menu”
   - Controls don’t have menus
   - So hMenu parameter used to hold control’s integer ID
   - ID value passed with WM_COMMAND message generated when user interacts with the control
   - Allows program to identify which control was activated

10. Handle to instance of program creating control
    - GetWindowLong() usually used to get this value

11. Pointer to window creation data
    - Normally NULL

Example (Win32 API)

- In response to WM_CREATE in Main Window’s WndProc():
  
  HWND hMyButton;
  HINSTANCE hInstance;
  hInstance = (HINSTANCE) GetWindowLong (hWnd, GWL_HINSTANCE);
  hMyButton = CreateWindow ("BUTTON", "Push Me", WS_CHILD | BS_PUSHBUTTON, 10, 10, 130, 60, hWnd, (HMENU)ID_MYBUTTON, hInstance, NULL);
  ShowWindow (hMyButton, SW_SHOWNORMAL);

Creating Controls -- MFC

- CWnd is the parent class of controls
- Define control in a related class or handler, eg.:
  
  CStatic myCtrl;
- Use the control’s override of CWnd::Create() to create the control (typically in OnCreate() handler)
  - Mostly same parameters as CreateWindow(), e.g.:
    
    RECT r;
    r.left = r.right = 10; r.right = 200; r.bottom = 30;
    myCtrl.Create ("Hello", WS_CHILD | WS_VISIBLE | SS_LEFT, r, this, ID_MYSTATIC);
  - Last parameter the control ID (defined in a .h file)

Using a Child Window Control, MFC

- Manipulate the control using its (and CWnd parent class) member functions
  - See Online help
- When finished with the control, use CWnd::DestroyWindow() to destroy the control

Messages from 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

Win32 API Control Message Handlers

- Put Control message handlers in same switch/case statement with menu handlers (WM_COMMAND)
  - Done just as for menu handlers
MFC Control Message Handlers
- Set up message macro for each notification code of interest
  - e.g., for button’s BN_CLICKED notification.
  - code:
    - ON_BN_CLICKED (ID, OnClickHandler)
- Declare the handler functions in the .h file
- Write the handler functions in .cpp file, e.g.
  ```cpp
  void CMyProgView::OnClickHandler()
  { // code goes here ;}
  ```

Sending Messages to Controls, Win32 API
- `SendMessage()`—sends message to a window’s WinProc()
- Doesn't return until message has been processed
- Parameters:
  - Handle of destination window
  - ID of message to send
  - wParam and lParam values containing message data, if any

Example, Win32 API
- Send a message to hMyControl
  ```cpp
  SendMessage (hMyControl, WM_SETTEXT, 0,
  (LPARAM) "Hello") ;
  ```
  - Here message is WM_SETTEXT
  - When received, control’s WinProc() changes control’s window name (text string displayed)
  - For this message wParam must be 0;
- There are many messages that can be sent to a control
- Depend on type of control. See online help

Sending Messages to Controls, MFC
- Use the Control’s `SendMessage()` function to send the control a message
- For example, assume m_myStatic is a CStatic object that has been created
- To change the text displayed:
  ```cpp
  char cBuf[] = “Hello”;
  m_myStatic.SendMessage (WM_SETTEXT, 0,
  (LPARAM)cBuf );
  ```

Alternatives to `SendMessage()`
- Could use other class member functions
- For most messages that can be sent to a control, there is a corresponding function
- Most are members of CWnd parent class
- Example sending WM_SETTEXT to a static control
  ```cpp
  m_myStatic.SetWindowText(“Hello”);
  ```
- Could also use `PostMessage()`
  - Returns immediately

Static Controls
- Lots of styles, see online help on “Static Control Styles”, Some examples:
  - SS_BITMAP, SS_CENTER, SS_GRAYFRAME, SS_ICON, SS_SIMPLE, SS_WHITEFRAME, etc.
- Change text with WM_SETTEXT message or SetWindowText()
  - May need to format values with `wsprintf()`
- Retrieve text with WM_GETTEXT message or GetWindowText()
- Program examples: static, static_mfc
Button Controls
- Some Styles: BS_PUSHBUTTON, BS_RADIOBUTTON, BS_CHECKBOX, BS_OWNERDRAW, BS_GROUPBOX, etc.
- Button notification codes:
  - BN_CLICK, BN_DOUBLECLICK
- Some messages you can send to buttons:
  - BM_SETCHECK, BM_GETCHECK, BM_SETSTATE, BM_GETSTATE, etc.
- Corresponding CButton member functions:
  - SetCheck(), GetCheck(), SetState(), GetState()
- Program examples: button, button_mfc

Graphical Push Buttons
- One way: use CBitmapButton class
- Assume we have a CBitmapButton object called m_bitmapbut and two bitmaps in the resources:
  - IDB_BMUP: “up state” bitmap
  - IDB_BMDOWN: “down state” bitmap
- Some code:
  m_bitmapbut.Create(“”, WS_CHILD | WS_VISIBLE | BS_OWNERDRAW, rect, this, BITMAP_BUTTON);
  m_bitmapbut.LoadBitmaps(IDB_BMUP, IDB_BMDOWN, 0, 0);
- Program Example: button_bitmap_mfc

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