Dialog Boxes

Dialog Boxes

- Popup child windows created by Windows
- Used for special-purpose input & output
  - Principal I/O mechanism in Windows
- Contain several child window controls
- Layout & what it does is are predefined (template – a resource)
- How it does is determined by a "Dialog box procedure"
- Destroyed immediately after use

Rule of Thumb: Dialog Boxes vs. programmer-defined child windows

- Dialog box: For simple popup windows that use normal window controls and do little painting on the client area
- Popup/child windows: Use when extensive painting or nonstandard behavior needed
- Main advantage of dialog boxes:
  - Ease of construction with the dialog box editor
  - Ease of communicating with its controls

Steps in Using:

- 1. Set up the template in the resources (.rc file)
  - Specifies controls used, their style/layout
  - Can be prepared "visually" with Visual Studio dialog box editor
  - Or "manually" with a text editor

- 2. (Win32 API) Write the dialog box procedure
  - Code to carry out dialog box's tasks
  - Placed in .cpp file
  - Provides message-processing capability
  - Messages from controls handled inside this procedure
  - Messages can be sent to the dialog box
  - A callback function like main window procedure *WndProc()*
  - But not the same
    - Part of the callback is inside Windows
    - It interprets some keystrokes (tab)
    - It calls our procedure
- 2. (MFC) Instantiate a CDialog object

Types of Dialog Boxes

- Modal
- Modeless
- System Modal

Modal

- While visible, user can't switch back to parent window
  - (Can change to other apps)
- User must explicitly end dialog box
  - Typically by clicking "OK" or "CANCEL" buttons inside
- Most common type of dialog box
- Example: "About" box available with most Windows apps
- Message Boxes are simple Modal Dialog Boxes

System Modal

- A variety of modal dialog box
- With these user can't switch to other applications while dialog box is active
- A throwback to Win16

Modeless

- User can switch between dialog box and the parent window
- More like popup windows
- Used when dialog box must be visible while user interacts with the parent
- Example: dialog box resulting from "Find" or "Replace" menu item of many Windows apps

Steps in Designing, Creating, Using a Modal Dialog Box: Win32 API

1. Determine child window controls needed inside
2. Design dialog box template (easiest with dialog box editor)
3. Write message-processing function
4. Activate dialog box by calling *DialogBox()*
   - Typically in response to menu item selection in *WndProc()*
5. Resulting dialog box stays on screen until call to *EndDialog()*
   - from inside dialog box function

DialogBox()

- Parameters:
  - 1. App's instance handle
  - 2. Dialog box ID name
    - Specified in dialog box template when .rc file created
  - 3. Handle of dialog box's parent window
  - 4. Address of *dialog box function* that will process its messages
    - A callback function
- Creates modal dialog box from app's dialog box template resources
- Displays dialog box & switches msg-processing to it
- Control returned when its msg-processing function terminates dialog box
  - By calling *EndDialog()*,

WM_INITDIALOG Message

- Like ordinary an window's WM_CREATE message
- Processed before window (dialog box) is made visible
- Good place to put dialog box initialization code

EndDialog()

- **Destroys dialog box**
- **Returns control to function (*WndProc()*) that called *DialogBox()***
- **Parameters:**
- **1. window handle passed to dialog box function (hDlg)**
- **2. integer value returned by *DialogBox()***
  - **Way of getting info from dialog box function to calling program**

User Interaction with Dialog Box Controls

- **WM_COMMAND message**
  - **LOWORD(wParam) contains control ID (as usual)**
  - **lParam, wParam contain message data (as usual)**

Exchanging Data with a Dialog Box

- **Exchanging data between dialog box function and app's *WndProc()***
- ***SendMessage()* could be used to send message to control inside, BUT:**
  - **Need to know control's handle**
  - **Not known since Windows creates the controls**
  - **IDs are known--specified in resource template**
- **Use *GetDlgItem()* to get control's handle:**
  - ***hControl = GetDlgItem(hDlg, controlID);***
- **Then *SendMessage(hControl, Msg, wParam, lParam);***

- **Both functions can be combined using *SendDlgItemMessage():***
- ***SendDlgItemMessage(hDlg, controlID, Msg, wParam, lParam);***
  - **Sends Msg to control whose ID is controlID**

Using Modal Dialog Boxes in MFC

- Dialog boxes are encapsulated by CDialog class (derived from CWnd)
- 1. App derives its own dialog box from CDialog
  - E.g., CSampleDlg : public CDialog
  - Constructor specifies that the parent constructor is to be used
  - Dialog box msg handling done w/ message maps
  - Dialog box class declarations (.h file):
    - Message handling functions
    - Message map declaration
  - Dialog box class implementation (.cpp file) defines:
    - Dialog box message map
    - Message handler function definitions

- 2. Creating the Dialog Box:
  - Instantiate a Dialog class object
  - Constructor of CDialog-derived class should call CDialog constructor
    - Arguments: ID of dialog box (specified in .rc file), pointer to owner window
      - CSampleDlg::CSampleDlg(CWnd* pParent) : Cdialog(CSampleDlg::IDD, pParent)
    - Creates the dialog box (not activated yet)
    - Initialization code should be put in CDialog's OnInitDialog() handler function
      - Invoked in response to WM_INITDIALOG message

- 3. Activating the Dialog Box
  - Use CDialog's DoModal() member function
  - Displays the dialog box
  - Messages from dialog box controls go to dialog box handler functions
  - Continues until dialog box has been closed
    - Use CDialog's EndDialog() member function
    - Causes DoModal() to return
    - Message processing continues in parent window

Communicating with Dialog Box Controls (exchanging data)

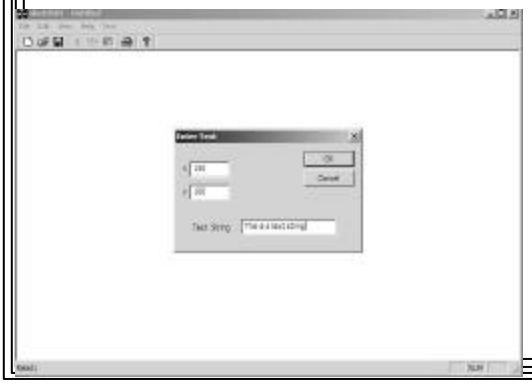
- Method 1
  - Get a pointer to control's ID w/ CWnd::GetDlgItem()
  - Use pointer to send appropriate messages to control, e.g. (for a list box in a dialog box):
    - CLISTBOX* plistbox=(CListBox*)GetDlgItem(IDD_XXX);
    - Plistbox->member_function();
  - OK for non-Wizard-generated apps
  - There's a much easier way for Wizard-generated apps

- Method 2
  - Automatically built into Wizard-generated Apps
  - Use DDX (Dynamic Data Exchange) mechanism
  - DDX system moves data between dialog box controls and variables in CDialog class
  - Occurs when a call is made to CWnd::UpdateData(direction);
  - Boolean parameter sets direction of data movement
    - TRUE ✍ from controls to variables
    - FALSE ✍ from variables to controls

- MFC's CDialog::OnInitDialog() calls UpdateData(FALSE) automatically
  - (Recall that this is called by your app to start the dialog box)
    - So Data from program variables is transferred automatically to dialog box controls when the dialog box starts
- MFC's CDialog::OnOK() calls UpdateData(TRUE)
  - (This is called when user clicks "OK" button inside dialog box)
    - So data from dialog box controls is transferred automatically to program variables when user clicks the dialog box's "OK button)
    - OnOK() then calls CDialog::EndDialog()
      - So dialog box disappears and DoModal() returns
        - Returns IDOK or IDCANCEL depending on user action
      - Destructor destroys the dialog box

Adding a Modal Dialog Box to the Sketching MFC Application

- **Will allow the user to specify where the text is to be displayed**
- **And what the text will be**



- Create a new Visual C++, MFC, SDI application (as usual)
- Add the sketching code (see earlier example)
- Add a new "Text" menu item (ID_TEXT)
- Add the new dialog box
  - Project/Add Resource/Dialog/New
  - Change ID to IDD_TEXT
  - Caption: "Enter Text"
- Use the dialog box editor to drag over 3 static and 3 edit controls:
  - Static Controls: "x", "y", "Text String"
  - Edit controls: IDC_X, IDC_Y, IDC_TESTEDIT

- Create the new Dialog Class
  - Right click on an unoccupied area of the dialog box & choose "Add Class" to bring up the "MFC Class Wizard" Dialog Box
  - Class name: "CTextDlg"
  - Base class: "Cdialog"

- Add New Class Variables (and connect to controls)
  - In Class View, right click on CTextDlg & choose Add variable
    - In resulting "Add member variable Wizard"
      - Check "Control Variable" check box
      - Control ID: IDC_X
      - Category: Value
      - Variable type: UINT
      - Variable name: m_x
  - In same way, attach UINT m_y to IDC_Y
  - And Cstring m_text to IDC_TEXTEDIT

- Add handler code to newCView "Text" menu item
  - In Class View select CView-derived class
  - In Properties Wizard Box "Events" (lightning bolt icon):
    - Scroll down to ID_TEXT
    - Add Command handler OnText()
    - Edit the resulting code by adding:

```
CTextDlg dlg;
Dlg.DoModal();
pDC = GetDC();  (Assumes a CDC* pDC variable)
pDC -> TextOut(dlg.m_x, dlg.m_y, dlg.m_text,
   lstrlen(dlg.m_text) );
```

- At top of Cview .cpp file underneath the other include statements, add:
- #include TextDlg.h