Windows Programming with MFC

MFC Programming

- MFC: The Microsoft Foundation Class Library
- Additional Notes:
  - http://www.cs.binghamton.edu/~reckert/360/class15.htm
MFC

The Microsoft Foundation Class (MFC) Library--
- A Hierarchy of C++ classes designed to facilitate Windows programming
- An alternative to using Win32 API functions
- A Visual C++ Windows app can use either Win32 API, MFC, or both

The Relationship between Windows
MFC and Win32 API Programming
Some Characteristics of MFC

- Offers convenience of REUSABLE CODE
  - Many tasks in Windows apps are provided by MFC
  - Programs can inherit and modify this functionality as needed
  - MFC handles many clerical details in Windows pgms
  - Functionality encapsulated in MFC Classes
- Produce smaller executables
- Can lead to faster program development
- MFC Programs must be written in C++ and require the use of classes
  - Programmer must have good grasp of OO concepts

Help on MFC Classes

- See Online Help (Index) on:
  "MFC"
  "Hierarchy"
  "Hierarchy Chart"
  "MFC Reference"
- On the Web:
Base MFC Class

- **CObject**: At top of hierarchy ("Mother" of almost all MFC classes)
- Provides features like:
  - Serialization
  - Runtime class information
  - Diagnostic & Debugging support
  - Some important macros
- All its functionality is inherited by any classes derived from it

Some Important Derived Classes

- **CFile**
- **CDC**
- **CGdiObject**
- **CMenu**


**CCmdTarget**: Encapsulates message passing process and is parent of:

- **CWnd**
  - Base class from which all windows are derived
  - Encapsulates many important windows functions and data members
  - Examples:
    - `m_hWnd` stores the window’s handle
    - `Create(...)` creates a window

- Most common subclasses:
  - `CFrameWindow`
  - `CView`
  - `CDialog`

**CCmdTarget** also parent of:

- **CWinThread**: Defines a thread of execution and is the parent of:
  - **CWinApp**
    - Encapsulates an MFC application
    - Controls following aspects of Windows programs:
      - Startup, initialization, execution, the message loop, shutdown
      - An application should have one CWinApp object
      - When instantiated, application begins to run

- **CDocument**
Primary task in writing an MFC program

- To create/modify classes
  - Most will be derived from MFC library classes
- Call class functions to perform tasks

MFC Class Member Functions

- Most functions called by an application will be members of an MFC class
- Examples:
  - `ShowWindow()`—a member of `CWnd` class
  - `TextOut()`—a member of `CDC`
  - `LoadBitmap()`—a member of `CBitmap`
- Applications can also call API functions directly
  - Use “global scope resolution” operator `::`
    - Example: `::UpdateWindow(hWnd);`
MFC Global Functions

- Not members of any MFC class
- Independent of or span MFC class hierarchy
- Example:
  - AfxMessageBox()

Message Processing under MFC

- API mechanism: switch/case statement in app’s WndProc
- Under MFC, WndProc is buried in MFC framework
- Message handling mechanism: “Message Maps”
  - lookup tables the MFC WndProc searches
- A Message Map contains:
  - A Message number
  - A Pointer to a message-processing function
    - These are members of CWnd
    - You override the ones you want your app to respond to
    - Like virtual functions
  - “Message-mapping macros” set these up
Simplest MFC programs must contain two classes derived from the hierarchy:
- 1. An application class derived from `CWinApp`
  - Defines the application
  - Provides the message loop
- 2. A window class usually derived from `CWnd` or `CFrameWnd`
  - Defines the application's main window

To use these & other MFC classes you must have:
`#include <Afxwin.h>` in the .cpp file
MFC Windows Programming (Document/View Approach)

- Frequently need to have different views of same data
- Doc/View approach achieves this separation:
  - Encapsulates data in a CDocument class object
  - Encapsulates data display mechanism & user interaction in a CView class object

Relationship between Documents, Views, and Windows
Document/View Programs

Almost always have at least four classes derived from:
- CFrameWnd
- CDocument
- CView
- CWinApp

Usually put into separate declaration (.h) and implementation (.cpp) files

Lots of initialization code

Could be done by hand, but nobody does it that way

Microsoft Developer Studio
AppWizard and ClassWizard Tools
AppWizard

- Tool that generates a Doc/View MFC program framework automatically
- Can be built on and customized by programmer
- Fast, efficient way of producing Windows Apps
- Creates functional `CFrameWnd`, `CView`, `CDocument`, `CWinApp` classes
- After AppWizard does it's thing:
  - Application can be built and run
  - Full-fledged window with all common menu items, tools, etc.

Other Visual Studio Wizards

- Dialog boxes that assist in generating code
  - Generate skeleton message handler functions
    - Set up the message map
  - Connect resources & user-generated events to program response code
  - Insert code into appropriate places in program
    - Code then can then be customized by hand
  - Create new classes or derive classes from MFC base classes
  - Add new member variables/functions to classes
- In .NET many wizards available through ‘Properties window’
SKETCH Application

- Example of Using AppWizard and ClassWizard
- User can use mouse as a drawing pencil
  - Left mouse button down:
    - lines in window follow mouse motion
  - Left mouse button up:
    - sketching stops
- User clicks "Clear" menu item
  - window client area is erased

- Sketch data (points) won’t be saved
  - So leave document (CSketchDoc) class created by AppWizard alone
- Base functionality of application (CSketchApp) and frame window (CMainFrame) classes are adequate
  - Leave them alone
- Use ClassWizard to add sketching to CSketchView class
Sketching Requirements

Each time mouse moves:
- If left mouse button is down:
  - Get a DC
  - Create a pen of drawing color
  - Select pen into DC
  - Move to old point
  - Draw a line to the new point
  - Make current point the old point
  - Select pen out of DC

Variables

BOOLEAN m_butdn
CPoint m_pt, m_ptold
COLORREF m_color
CDC* pDC
Steps in Preparing SKETCH

1. “File” / “New” / “Project”
   – Project Type: “Visual C++ Projects”
   – Template: “MFC Application”
   – Enter name: Sketch

2. In “Welcome to MFC Application Wizard”
   – Application type: “Single Document” Application
   – Take defaults for all other screens

3. Build Application --> Full-fledged SDI App with empty window and no functionality

4. Add member variables to CSketchView
   – Can do manually in .h file
   – Easier to:
     • Select Class View pane
     • Click on SketchView class
       – Note member functions & variables
     • Right click on CSketchView class
       – Choose “Add / Variable”
         – Launches “Add Member Variable Wizard”
       – Variable Type: enter CPoint
       – Name: m_pt
       – Access: Public (default)
         – Note after “Finish” that it's been added to the .h file
     • Repeat for other variables (or add directly in .h file):
       – CPoint m_ptold
       – bool m_butdn
       – COLORREF m_color
       – CDC* pDC
5. Add message handler functions:
   - Select CSketchView in Class View
   - Select “Messages” icon in Properties window
     • Results in a list of WM_ messages
   - Scroll to WM_LBUTTONDOWN & select it
   - Add the handler by clicking on down arrow and
     “<Add> OnLButtonDown”
     • Note that the function is added in the edit window and the
       cursor is positioned over it:
       - After “TODO…” enter following code:
         ```
         m_butdn = TRUE;
         m_ptold = point;
         ```
Repeat process for WM_LBUTTONDOWN handler:

- Scroll to WM_LBUTTONDOWN
- Click: "<Add> OnLButtonDown",
- Edit Code by adding:
  
m_butdn = FALSE;
Repeat for WM_MOUSEMOVE
   - Scroll to WM_MOUSEMOVE
   - Click: "<Add> OnMouseMove"
   - Edit by adding code:
     if (m_butdn)
     {
         pDC = GetDC();
         m_pt = point;
         CPen newPen (PS_SOLID, 1, m_color);
         CPen* pPenOld = pDC->SelectObject(&newPen);
         pDC->MoveTo (m_ptold);
         pDC->LineTo (m_pt);
         m_ptold = m_pt;
         pDC->SelectObject (pPenOld);
     }

6. Initialize variables in CSketchView constructor
   - Double click on CSketchView constructor
     • CSketchView(void) in Class View
   - After "TODO...", Add code:
     m_butdn = FALSE;
     m_pt = m_ptold = CPoint(0,0);
     m_color = RGB(0,0,0);
7. Changing Window’s Properties
   – Use window’s SetWindowXxxxx() functions
     • In CWinApp-derived class in its InitInstance(…) function before window is shown and updated
   – Example: Changing the default window title
     \[ m_pMainWnd->SetWindowTextW( 
      TEXT("Sketching Application")); \]
   – There are many other SetWindowXxxxx() functions that can be used to change other properties of the window

8. Build and run the application

Menu and Command Messages

- User clicks on menu item
- WM_COMMAND message is sent
- ID_XXX identifies which menu item (its ID)
- No predefined handlers
  - We write the OnXxx() handler function
  - Must be declared in .h file and defined in .cpp file
- Event handler wizard facilitates this
Adding Color and Clear Menu Items to SKETCH App

Resource View (sketch.rc folder)
- Double click Menu folder
- Double click IDR_MAINFRAME menu
- Add: “Drawing Color” popup menu item with items:
  - “Red”, ID_DRAWINGCOLOR_RED (default ID)
  - “Blue”, ID_DRAWINGCOLOR_BLUE
  - “Green”, ID_DRAWINGCOLOR_GREEN
  - “Black”, ID_DRAWINGCOLOR_BLACK
- Add another main menu item:
  - “Clear Screen”, ID_CLEARSCREEN
    - Set Popup property to False

Add Menu Item Command Handler Function

One way: Use “Event Handler Wizard”
- In “Resource View” bring up menu editor
- Right click on “Red” menu item
- Select “Add Event Handler” “Event Handler Wizard” dialog box
  - Class list: CSketchView
  - Message type: COMMAND
  - Function handler name: accept default
    - OnDrawingcolorRed
  - Click on “Add and edit”
  - After “TODO…” in editor enter following code:
    m_color = RGB(255,0,0);
Another Method of Adding a Menu Item Command Handler

- In Class View Select CSketchView
- In Properties window select Events (lightning bolt icon)
- Scroll down to: ID_DRAWINGCOLOR_RED
- Select “COMMAND”
- Click “<Add> OnDrawingcolorRed” handler
- Edit code by adding:
  
  ```
  m_color = RGB(255,0,0);
  ```
Repeat for ID_DRAWINGCOLOR_BLUE
Code: m_color = RGB(0,0,255);
Repeat for ID_DRAWINGCOLOR_GREEN
Code: m_color = RGB(0,255,0);
Repeat for ID_DRAWINGCOLOR_BLACK
Code: m_color = RGB(0,0,0);
Repeat for ID_CLEAR
Code: Invalidate();
Destroying the Window

- Just need to call $\texttt{DestroyWindow()}$
  - Do this in the CMainFrame class – usually in response to a “Quit” menu item

Build and Run the Application