Introduction to Microsoft Windows MFC Programming: The Application/Window Approach

Additional notes at:
www.cs.binghamton.edu/~reckert/360/class14.htm

MFC Windows Programming
- 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 application can use either Win32 API, MFC, or both

Microsoft Foundation Classes
- About 200 MFC classes (versus 2000+ API functions)
- Provide a framework upon which to build Windows applications
- Encapsulate most of the Win32 API in a set of logically organized classes

Some characteristics of MFC
- 1. Convenience of reusable code:
  - Many tasks common to all Windows apps are provided by MFC
  - Our programs can inherit and modify this functionality as needed
  - We don't need to recreate these tasks
  - MFC handles many clerical details in Windows programs
    - e.g., WinMain, WndProc, and message loop are buried in the MFC Framework

MFC Characteristics, continued
- 2. Produce smaller executables:
  - Typically 1/3 the size of their API counterparts
- 3. Can lead to faster program development:
  - But there's a steep learning curve--
  - Especially for newcomers to object-oriented programming
MFC Characteristics, continued

4. MFC Programs must be written in C++ and require the use of classes
   - Programmer must have good grasp of:
     • How classes are declared, implemented (instantiated), extended, overridden, and used
     • Encapsulation
     • Inheritance
     • Polymorphism
     - Virtual functions

Help on MFC Classes
- See Online Help (Index) on:
  “MFC” | “classes”
  “MFC classes (MFC)”
- Clicking on a class a document with a link to the class members
- Also look at
  “MFC” | “hierarchy”
  “hierarchy chart”

Base MFC Class
- CObject: At top of hierarchy ("Mother of almost all MFC classes")
- Provides features like:
  - Serialization
    • Streaming an object’s persistent data to or from a storage medium (disk reading/writing)
  - 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: Support for file operations
- CArchive: Works with CFile to facilitate serialization and file I/O
- CDC: Encapsulates the device context (Graphical Drawing)
- CGdiObject: Base class for various drawing objects (CBrush, CPen, CFont, etc.)
- CMenu: Encapsulates menus and menu management

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: Can contain other windows
  - ("normal" kind of window we’ve used)
  • CView: Encapsulates process of displaying and interacting with data in a window
  • CDialog: Encapsulates dialog boxes

CCmdTarget also the parent of:
- CWinThread: Defines a thread of execution
  - m_pMainWnd is a member of this class
  - A pointer to an application’s main window
  - Is the parent of:
    • CWinApp: Most important class dealt with in MFC applications
      - 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
        - Member function InitInstance() is called by WinMain()
    • m_nCmdShow is a member of this class
  - CDocument
    - Encapsulates the data associated with a program
MFC Classes and Functions
- Primary task in writing MFC program—to create classes
- Most will be derived from MFC library classes
- Encapsulate 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 class
  - `LoadBitmap()` -- a member of CBitmap class

Applications can also call API functions directly
- Use Global Scope Resolution Operator (::), for example:
  - `::UpdateWindow(hWnd);`
- Usually more convenient to use MFC member functions

MFC Global Functions
- Not members of any MFC class
- Begin with Afx prefix (Application Framework)
- Independent of or span MFC class hierarchy
- Example:
  - `AfxMessageBox()` • Message boxes are predefined windows
  - Can be activated independently from the rest of an application
  - Good for debugging

Some Important Global Functions
- `AfxAbort()` -- Unconditionally terminate an app
- `AfxBeginThread()` -- Create & run a new thread
- `AfxGetApp()` -- Returns a pointer to the application object
- `AfxGetMainWnd()` -- Returns a pointer to application's main window
- `AfxGetInstanceHandle()` -- Returns handle to application's current instance
- `AfxRegisterWndClass()` -- Register a custom WNDCLASS for an MFC app

A Minimal MFC Program (App/Window Approach)
- 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

Message Processing under MFC
- Like API programs, MFC programs must handle messages from Windows
- API mechanism: switch/case statement in app’s `WndProc()`
- In MFC, `WndProc()` is buried in the MFC library
- Message handling mechanism: “Message Maps”
  - Lookup tables the MFC `WndProc()` searches
- Table entries:
  - Message number
  - Pointer to a message-processing function
    - These functions are members of `CWnd`
    - We override the ones we want our program to respond to
Message Mapping

- Programs must:
  - Declare message-processing (handler) functions
    - e.g., `OnWhatever()` for `WM_WHATEVER` message
  - Map them to messages program is going to respond to
    - Mapping done by "message-mapping macros"
    - e.g., `ON_WM_WHATEVER()`

---

STEPS IN WRITING A SIMPLE MFC PROGRAM
(App/Window Approach)

1. Declare a window class derived from `CFrameWnd` (e.g., `CMainWin`—

   - Class Members:
     - The constructor declaration
     - Message-processing function declarations for messages
       the application will override and respond to
       - e.g., `void OnChar(...)`
     - `DECLARE_MESSAGE_MAP()` macro:
       - Allows windows based on this class to respond to messages
       - Declares that a message map will be used to map messages to
       overruling handler functions in the application
       - Should be last class member declared

2. Declare an application class derived from `CWinApp` (e.g., `CApp`—

   - Must override `CWinApp’s InitInstance()` virtual function:
     - Called each time a new instance of application is started
     - i.e., when an object of this application class is instantiated
     - Purpose is for application to initialize itself
     - Good place to put code that does stuff that has to be
done each time program starts

---

DECLARATION (.h)

1. Declare a window class derived from `CFrameWnd` (e.g., `CMainWin`—

2. Declare an application class derived from `CWinApp` (e.g., `CApp`—

   - Must override `CWinApp’s InitInstance()` virtual function:
     - Called each time a new instance of application is started
     - i.e., when an object of this application class is instantiated
     - Purpose is for application to initialize itself
     - Good place to put code that does stuff that has to be
done each time program starts

---

IMPLEMENTATION (.CPP)

1. Define constructor for class derived from `CFrameWnd` (e.g., `CMainWin`)

   - Should call member function `Create()` to create the window
     - Does what `CreateWindow()` does in API

2. Define message map for class derived from `CFrameWnd` (our `CMainWin`—

   - List of "message-mapping macros", e.g.,
     - `ON_WM_CHAR()`
3. Define (implement) message-processing functions declared in .h file declarations above

4. Define (implement) `InitInstance()` overriding function--
   - Done in class derived from `CWinApp ... our CApp`:
     - Should have initialization code:
       - Instantiate a `CMainWin` object (pointer to program's main window
       - `m_pMainWnd` (Used to refer to the window, like `hWnd` in API programs)
       - Invoke object's `ShowWindow()` member function
       - Invoke object's `UpdateWindow()` member function
       - Must return non-zero to indicate success
     - [MFC's implementation of WinMain() calls this function]

5. Instantiate the app class (e.g., `our CApp`)
   - Causes `AfxWinMain()` to execute
     - It's now part of MFC [WINMAIN.CPP]
   - `AfxWinMain()` does the following:
     - 1. Calls `AfxWinInit()`--
       - which calls `AfxRegisterClass()` to register window class
     - 2. Calls `CApp::InitInstance()` [virtual function overridden in 4 above]--
       - which creates, shows, and updates the window
     - 3. Calls `CWinApp::Run()` [In THRDCORE.CPP]--
       - which calls `CWinThread::PumpMessage()`--
       - which contains the `GetMessage()` loop

 MSG2005 Example MFC Application: Mouse/Character Message Processing

   - User presses mouse button
     - “L” or “R” displayed at current mouse cursor position
   - Keyboard key pressed
     - Character displayed at upper left hand corner of client area

   - Now nature & form of simple window & application have been defined
   - But neither exists --
   - Must instantiate an application object derived from `CWinApp ... our CApp`

   - After `CWinApp::Run()` returns:
     - (i.e., when the WM_QUIT message is received)
     - `AfxWinTerm()` is called--
     - which cleans up and exits

   - Message map contains:
     - `ON_WM_CHAR()`
     - `ON_WM_LBUTTONDOWN()`
     - `ON_WM_RBUTTONDOWN()`
   - To respond to messages:
     - `WM_CHAR`
     - `WM_LBUTTONDOWN`
     - `WM_RBUTTONDOWN`
   - So we need to define the following handler function overrides:
     - `CWnd::OnChar(UINT ch, UINT count, UINT flags)`;
     - `CWnd::OnLButtonDown(UINT flags, CPoint loc)`;
     - `CWnd::OnRButtonDown(UINT flags, CPoint loc)`;
In each handler we need to get a Device Context to draw on:

\[ CDC^* \text{pDC} \]

- Declare a pointer to a CDC object
- Use \text{GetDC()} member function of ‘this’ \text{CWnd} to get a device context to draw on

And then display a string using \text{TextOut()}:

- If it’s a character, it must be formatted into a string first
- Can use \text{wsprintf()}
  - Formats integers, characters, and other data types into a string

Steps in Creating and Building an MFC Application like msg2005 “manually”

1. “File” | “New” | “Project”
   - Specify an empty Win32 project as in previous examples
2. “Project” | “Add New Item”
   - Templates: “C++ File”
   - Enter or copy/paste .cpp file text (e.g., msg2005.CPP) – see \text{IMPLEMENTATION} above
   - Enter or copy/paste .h file text (e.g., msg2005.h) – see \text{DECLARATION} above
4. With project name highlighted in Solution Explorer window,
   - “Project” | “Properties” | “Configuration Properties” | “General”
   - From “Use of MFC”, choose:
     - “Use MFC in a Shared DLL”
      Build the project as usual

How It Works

\text{CApp} \text{ object is created}

- MFC’s \text{WinMain()} executes
- Registers class (default)
- Calls our \text{CApp::InitInstance()}
  - Our override creates a \text{CMainWin} object
  - Our \text{CMainWin} constructor calls \text{Create()} \text{ window is created}
  - Our \text{CApp::InitInstance()} override calls window’s \text{ShowWindow()} \text{ window is displayed}
  - Our override calls \text{UpdateWindow()} \text{ client area painted}
  - \text{WinMain()} continues by calling its \text{Run()} function
  - Call to \text{PumpMessage()}
  - Which starts the message loop
  - Message map sends messages of interest to our handler functions