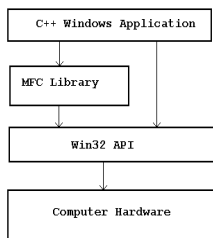


Introduction to Microsoft Windows MFC Programming: The Application/Window Approach

- 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



The Relationship between Windows MFC and Win32 API Programming

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

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

Help on MFC Classes

- ≈ See Online Help (Index) on:
 - “MFC (Microsoft Foundation Class)”
 - “Hierarchy Chart”
 - “Hierarchy Chart”
- Each class name is a hot link
- At bottom each has a “Class Members” link
 - Clicking ≈ a document that lists all class members
- ≈ On the Web:
 - http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vcmfc98/html/_mfc_class_library_reference_introduction.asp

Base MFC Class

- ≈ **CObject**: At top of hierarchy (“Mother of almost all MFC classes”)
- ≈ Provides features like:
 - Serialization
 - Streaming 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 menu management

≈ **CCommandTarget**: 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
 - **CDialog**: Encapsulates dialog boxes

≈ **CCommandTarget** also parent of:

- **CWinThread**: Defines a thread of execution and 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
- **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
- ≠ **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

- ≠ Apps 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 (**A**pplication **F**ramework**K**S)
- ≠ Independent of or span MFC class hierarchy
- ≠ Example:
 - *AfxMessageBox()*
 - Message boxes are predefined windows
 - Can be activated independently from the rest of an application

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 applications's current instance
- ≠ *AfxRegisterWndClass()* -- Register a custom WNDCLASS for an MFC app

A Minimal MFC Program (App/Window Approach)

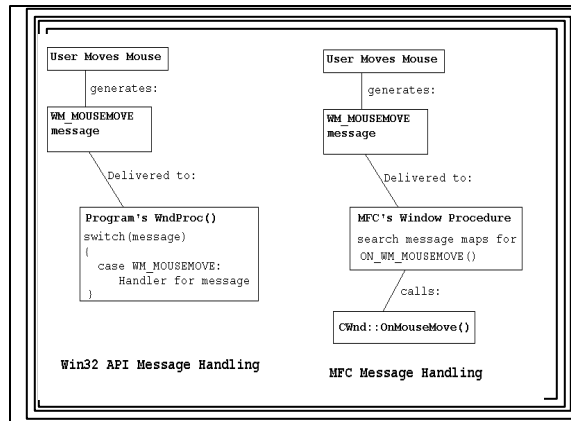
- ≠ Simplest MFC programs must contain two classes derived from the hierarchy:
 - 1. An application class derived from *CWinApp*
 - Defines the application
 - provides the message loop
 - 2. A window class usually derived from *CFrameWnd*
 - Defines the application's main window
- ≠ To use these & other 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
 - ≠ Under MFC, WndProc is buried in MFC framework
 - ≠ Message handling mechanism: “**Message Maps**”
 - lookup tables the MFC WndProc searches
 - ≠ Table entries:
 - Message number
 - 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

- ⚡ Programs must:
 - Declare message-processing (handler) functions
 - e.g., `OnWhatever()` for `WM_WHATEVER` message
 - Map them to messages app is going to respond to
 - Mapping done by "message-mapping macros"
 - Bind a message to a handler function
 - e.g., `ON_WM_WHATEVER()`
- ⚡ Most MFC application windows use a window procedure, `WndProc()`, supplied by the library
- ⚡ Message maps enable library window procedure to find the function corresponding to the currentmsg



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

DECLARATION (.h)

1. Declare a window class derived from *CFrameWnd* (e.g., *CMainWin*)--
- ⚡ Class Members:
 - The constructor
 - Message-processing function declarations for messages the application will 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 overriding 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

IMPLEMENTATION (.CPP)

1. Define constructor for class derived from *CFrameWnd* (e.g., our *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*)--


```
BEGIN_MESSAGE_MAP(owner, base)
    List of "message-mapping macros", e.g.
    ON_WM_CHAR()
END_MESSAGE_MAP()
```

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 for each new app instance:
 - Instantiate a *CMainWin* object ≈ pointer to program's main window
 - (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]

≈ Now nature & form of simple window & application have been defined

≈ But neither exists --

≈ Must instantiate an application object derived from *CWinApp ... our CApp*

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:

- Calls *AfxWinInit()*--
 - which calls *AfxRegisterClass()* to register window class
- Calls *CApp::InitInstance()* [virtual function overridden in 4 above]--
 - which creates, shows, and updates the window
- Calls *CWinApp::Run()* [In THRD CORE.CPP]--
 - which calls *CWinThread::PumpMessage()*--
 - which contains the *GetMessage()* loop

≈ After *CWinApp::Run()* returns:

- (i.e., when the WM_QUIT message is received)

- *AfxWinTerm()* is called--

- which cleans up and exits

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

≈ 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* pDC

- Declare a pointer to a CDC object

pDC = this->GetDC();

- Use GetDC() member function of 'this' CWnd to get a device context to draw on

≈ And then display a string using TextOut()

- If it's a character, it must be formatted into a string first
- Can use 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"
 - Categories: "Visual C++" | "Code"
 - Templates: "C++ File"
 - Enter or copy/paste .cpp file text (e.g., MSGNEW.CPP)--see IMPLEMENTATION above
3. "Project" | "Add New Item" | "Visual C++" | "code" | "Header File"
 - Enter or copy/paste .h file text (e.g., MSGNEW.H)--see DECLARATION above
4. With project name highlighted in Solution Explorer window, "Project" | "Configuration Properties" | "General"
 - From "Use of MFC", choose:
 - "Use MFC in a Shared DLL"
5. Build the project as usual

How It Works

CApp object is created ≈

MFC's *WinMain()* executes ≈

Registers class (default)

Calls our *CApp::InitInstance()* ≈

Our override creates a *CMainWin* object

Our *CMainWin* constructor calls *Create()* ≈ window created

Our *CApp::InitInstance()* override calls window's

ShowWindow() ≈ window is displayed

Our override calls *UpdateWindow()* ≈ client area painted

WinMain() continues by calling its *Run()* function ≈

Call to *PumpMessage()*

Which starts the message loop