An Introduction to MFC
(builds on your lecture on MFC)

Tutorial
(First Steps in Visual C++ MFC)
[For hands on experience go through sample ‘Scribble’
given in MSDN … I cannot write anything better than
that]

By
Zubair Malik
WHAT IS AN API (Application Programming Interface)

- The interface between my toaster and electric system is standard plug. No matter where I am I know how to use the plug to get a valuable service: electric power.

- Operating system of the computer is managing valuable services like: processes management, memory management, file system management and devices connected to PC etc.

- API is a standard interface defined to communicate with OS to gain access to these services.

- Your programs provide us an interface as well: The Public methods and variables of your classes that we know how to access. [They have exact name and parameters, else they don’t work … just like a plug has to be of a given size and shape].
WHY MFC

• API calls are long and cryptic
• The development time will be much high if we only base our program on those call.
• MFC is a wrapper for APIs (but does not cover all APIs)
• Also provides very useful classes to make development easy (just like STL).
• Understanding MFC requires minimum understanding of Windows OS Architecture. [process, virtual memory, thread, and multitasking]
The Architecture Basics

- application is a static sequence of instructions.
- process occurs when the application is invoked and consists of all the system resources

A process consists of:
- An executable program
- A private address space in memory
- System resources, such as files, pipes, communications ports, and semaphores
- At least one thread, where a thread is a path of execution

A processes consists of one or many threads.

(process:thread::program:function)
The Architecture Basics (continued …)

- **Physical memory**  The amount of physical RAM.
- **Virtual memory** Consists of 4GB of addresses, or $2^{32}$ bytes of addressable memory, that is available to your application.
- This is *not* 4GB of actual physical memory. Each application is given 2GB of addresses while the operating system reserves 2GB for its own use.
- The application never directly accesses physical memory
The Architecture Basics (continued …)

- Multitasking: an environment in which, from the user's point of view, multiple programs appear to execute simultaneously.
- A processor is capable of processing one thread at a time. Multitasking under Windows involves dividing the CPU time among the threads that belong to the various processes.
Now the Real thing
[build up was just to stress on what Mr. Irfan told us in the class]

Messages and Message Queues

- Windows environment is a message-based system
- Whenever an event of interest occurs a message is generated by Windows; How an application responds to these messages determines its behavior.[WE WILL BE WRITING EVENT HANDLERS]
- Messages contain information about the event, such as its type, the time it occurred, and the specific window to which it was directed.
- Each thread has its own virtual input queue for processing messages that are sent to it.
A windows Program

• A windows program has two main tasks (other than a normal program)
  – Preparing a window class to use with an application
  – Creating the message-handling structure of the application

• Basic elements therefore are:

• Code: As with any application, its primary element is executable code. Windows-based applications have an entry point called **WinMain**.

• Resources (Dialog box, Tool Bar, Icon ..)

• DLLs (a method for invoking library modules at runtime)
Message-handling structure

- From a programmer's standpoint, windows are the primary method of communicating information from the application to the user.
- When a Windows-based application is started, the program begins at the entry point (WinMain).
- The application then creates one or more windows.
- Each window contains a **window procedure** that is responsible for determining what the window displays and how the window responds to user input.
- A **message loop** retrieves messages from the **message queue** and gives them back to Windows to send to the appropriate window procedure.
Types of Windows

• the layering of the windows is managed by the Windows operating system (layering is called the “Z order.”)

• Top level window is desktop and managed by OS, all other are its children

• An application has:
  – at least a single top-level window
  – the main application window

• Windows can have parent-child relationship or owner-owned relationship.
Drawing and Device Context (DC)

• Instead of the application handling the particulars of output to a device, Windows provides device independence through the use of a DC.

• DCs:
  – are logical constructs that are used to maintain the integrity of output
  – Give permission to an application to use an output device
  – Provide a link between a Windows-based application, a device driver, and an output device such as a monitor or printer.
  – Maintain current information about how to draw or paint to a window, such as the colors, brush or pen patterns, pen widths, and so on.
Painting

• Windows may obscure other windows
• Windows is responsible for maintaining the Z order, your application is responsible for painting the information that appears in its client area
• Windows typically handles painting for nonclient areas such as the frame, scroll bars, the title bar, and the window menu.
• o restore information lost from the client area, you must write a handler for the WM_PAINT message
• A repaint operation is only allowed within the current invalid region.
Classification of MFC

• MFC is based on inheritance (MS may claim it as best example of it)
• MFC contains more than 150 classes. Some of these classes are used directly, while others serve as base classes for your own classes.
• Can be loosely classified into 7 parts
• **CObject**-derived classes
  – Application architecture classes
  – Visible object classes
  – OLE classes
  – Database classes
  – General purpose classes
  – Global **Afx** functions
• Please do not quote me on this information, I worked on MFC a long time ago … com dcom and .NET are in and I have not a good idea of them
• **CObject** is the base class for most classes in the remaining categories.