Microsoft Foundation Class Library

The Microsoft Foundation Class Library (also Microsoft Foundation Classes or MFC) is a library that wraps portions of the Windows API in C++ classes, including functionality that enables them to use a default application framework. Classes are defined for many of the handle-managed Windows objects and also for predefined windows and common controls.

MFC was introduced in 1992 with Microsoft's C/C++ 7.0 compiler for use with 16-bit versions of Windows as an extremely thin OOP/C++ wrapper for the Windows API. One interesting quirk of MFC is the use of "Afx" as the prefix for many functions, macros and the standard pre-compiled header name "stdafx.h".

MFC encapsulate key window data structure. Many MFC classes have member functions with names that are identical to those of native API functions. The MFC library encapsulates all normal procedure- oriented windows functions and provides support for control bars, property sheet , ActiveX control and database support etc. Thus MFC makes windows application development easier.

MFC library is a collection of C++ classes. It provides as a Dynamic Linking Library (DLL) so your application has access to the classes in MFC. A DLL contains a executable function that are loaded into memory and are independent from any application libraries such as MFC are called Application framework, because they give the user a framework for an application. The MFC classes have been built using the OS's API function .Using MFC classes means that much of the programming has already done for you and you need to add only special features to the MFC code to create your application. To use MFC framework your application mist be written in C++.

MFC is designed to work with all available windows OS like Windows95, 98, NT etc. MFC applications can be built and run an any these OS's.

MFC and Windows OS Interaction.

Windows OS has 3 major components – USER , GDI and KERNEL.

USER - USER is a module of code that services input devices such as keyboard, mouse etc.. .

Kernel – Kernel is a module that services file management and internal memory management.

GDI – This GDI serves output to graphical devices such as screen, printers etc.

Collectively these three components are called API. These components interact with the MFC application. MFC application calls functions in the API. Each of the t3 API components are provided as DLL. An application can call functions in the DLL as though they were part of the application. The API DLL,s are normally found in windows OS directory.

C:\WINDOWS\SYSTEM

Files are user.exe, gdi.exe and kernal386.exe ---- in Win16

user.dll, gdui.dll and kernal32.dll ---- in Win32

MFC Application Framework

MFC is a library of built in classes that can be used or derived for various functionalities of your application. MFC contains some relationship with Win32. Using MFC library classes windows programming can be done without Win32 API. There is no explicit WinMain( ) function in the MFC application framework structure, but there is an underlining WinMain will be called by the classes of MFC library

Application Framework is a different programming structure. An application framework is a software framework that is used to implement the standard structure of an application for a specific operating system. Application frameworks became popular with the rise of the graphical user interface (GUI), since these tended to promote a standard structure for applications. It is also much simpler to create automatic GUI creation tools when a standard framework is used, since the underlying code structure of the application is known in advance. Object-oriented programming techniques are usually used to implement frameworks such that the unique parts of an application can simply inherit from pre-existing classes in the framework.

MFC applications (Advantages)

- It is a C++ interface to windows API
- It contains several general purpose (non- window specific) classes like
    - collection of classes for list, array
    - String class
    - Time,Time spand, Date classes
    - File access class
- Multiple Document Interface (MDI) application support
- Support menu items
- It support scrolling window
- support tool bar(back, go, next) and status bar.
- Automatic processing of a data entered in a dialog box
- ODBC connectivity
- WinSock and WinIner classes for TCP/IP communication
- Support classes for thread synchronization.

MFC and its type of classes

The classes in the MFC library is mainly classified into 4 categories.

1. General classes

    These classes provide things like string -handling classes and collection classes
2. Window API classes

    These classes provides a wrapper over the windows OS
3. Application Framework classes

    These classes handle large pices of the whole application such as message-pumping logic, printing as well as MFC's document/view architecture.
4. High- level abstraction

    It is for abstracting several OS extensions, including OLE, MAPI and WinSock.

Class structure Diagram [MFC Class Hierarchy]

Cobject class – The mother of all classes

MFC's root class is called Cobject class. It define and implements functionalities that most MFC classes need in order to work with other parts of the framework. CObject serves the root not only for library classes such as Cfiles or an CobList, but also for the classes that you write. when you are using this MFC classes, you should usually make sure that you have Cobject some ware in your class hierarchy. When you derive your class from Cobject, your class automatically gain the ability to add the following 4 basic services.

- Serialization support
- Run-time class information
- Object diagnostic output
- Compatibility with collection classes

Basic Services for CObject

1.Run-time class information (RTCI)

CObject Run-time class information (RTCI) feature lets lets the developer determine information about an object such as class name and parent at runtime. MFC maintain these information by the help of the CRuntime class. Application rarely use the Cruntime class directely, but it depents on macrossuch as DECLARE_DYNAMIC (to be embedded in the definition of class) and IMPLEMENT_ DYNAMIC (to be added to the implementation file). These macros add the runtime information to the class and enable the use of IsKindOf member function.

IsKindOf is used to test the objects relationship to a given class.

2. Dynamic Creation [Compatibility with collection classes]

To add Dynamic Creation support to your CObject derivative, then you must use the DECLARE_DYNCREATE and IMPLEMENT_DYNCREATE macro instead of DECLARE_DYNAMIC and IMPLEMENT_DYNAMIC macros. Once you add this you can create objects based on there CRuntime class information through the use of create object member function.

3.Persistence [Serialization]

Persistence is the ability to store object and restore their stages some times later. Using persistence it is very easy to reading and writing without having to worry about the format of the file you are writing to. To support serialization in you must use the DECLARE_SERIAL and IMPLEMENT _SERIAL macro instead of DECLARE _DYNAMIC and IMPMENT_DYNAMIC.

4. Run Time Object Diagnostics [Object diagnostic output]

All CObject derived MFC classes have 2 member functions

1. Dump member function

The Dump member function is that you can call to print out a C++'s object state at run time.

The Dump routine makes you of the CDumpContext helper class to output the debugging information.

2. Assert Valid member function

The Assert Valid member function in which the object checks its member validity at run time.

Any other object can call Assert Valid member function to verify that the object is in a safe state.

Both of these member function are use in the debug builder to provide advanced debugging facilities.

CWinApp: The Application Class

The main application class in MFC encapsulates the initialization, running, and termination of an application for the Windows operating system. An application built on the framework must have one and only one object of a class derived from CWinApp. This object is constructed before windows are created.
The base class from which you derive a Windows application object.

```
class CWinApp : public CwinThread
```

CWinApp is derived from CWinThread, which represents the main thread of execution for your application, which might have one or more threads**.**In recent versions of MFC, the InitInstance, Run, ExitInstance, and OnIdle member functions are actually in class CWinThread.

Like any program for the Windows operating system, your framework application has a **WinMain** function. In a framework application, however, you do not write **WinMain**. It is supplied by the class library and is called when the application starts up. **WinMain** performs standard services such as registering window classes. It then calls member functions of the application object to initialize and run the application. (You can customize **WinMain** by overriding the **CWinApp** member functions that **WinMain** calls.)

To initialize the application, **WinMain** calls your application object's `InitApplication` and `InitInstance` member functions. To run the application's message loop, **WinMain** calls the **Run** member function. On termination, **WinMain** calls the application object's `ExitInstance` member function.

Each application that uses the Microsoft Foundation classes can only contain one object derived from CWinApp. This object is constructed when other C++ global objects are constructed and is already available when Windows calls the WinMain function, which is supplied by the Microsoft Foundation Class Library. Declare your derived CWinApp object at the global level.

When you derive an application class from CWinApp, override the InitInstance member function to create your application's main window object.

In addition to the CWinApp member functions, the Microsoft Foundation Class Library provides the following global functions to access your CWinApp object and other global information:

- AfxGetApp Obtains a pointer to the CWinApp object.
- AfxGetInstanceHandle Obtains a handle to the current application instance.
- AfxGetResourceHandle Obtains a handle to the application's resources.
- AfxGetAppName Obtains a pointer to a string containing the application's name. Alternately, if you have a pointer to the **CWinApp** object, use **m_pszExeName** to get the application's name.

AfxGetApp

The pointer returned by this function can be used to access application information such as the main message-dispatch code or the topmost window.
Return Value
A pointer to the single **CWinApp** object for the application.

AfxGetInstanceHandle
This function allows you to retrieve the instance handle of the current application.

Return Value

An **HINSTANCE** to the current instance of the application. If called from within a DLL linked with the USRDLL version of MFC, an **HINSTANCE** to the DLL is returned.

Remarks

AfxGetInstanceHandle always returns the **HINSTANCE** of your executable file (.EXE) unless it is called from within a DLL linked with the USRDLL version of MFC. In this case, it returns an **HINSTANCE** to the DLL.

AfxGetResourceHandle

Use the **HINSTANCE** handle returned by this function to access the application's resources directly, for example, in calls to the Windows function **FindResource**.

Return Value

An **HINSTANCE** handle where the default resources of the application are loaded.

AfxGetAppName

The string returned by this function can be used for diagnostic messages or as a root for temporary string names.

Return Value

A null-terminated string containing the application's name.

AfxSetResourceHandle

Use this function to set the **HINSTANCE** handle that determines where the default resources of the application are loaded.

```
void AFXAPI AfxSetResourceHandle( HINSTANCE hInstResource );
```

Parameters

hInstResource

The instance or module handle to an .EXE or DLL file from which the application's resources are loaded.

AfxFindResourceHandle

Use **AfxFindResourceHandle** to walk the resource chain and locate a specific resource by resource ID and resource type.

```
HINSTANCE AFXAPI AfxFindResourceHandle(
   LPCTSTR lpszName,
   LPCTSTR lpszType
);
```

Parameters

lpszName

A pointer to a string containing the resource ID.

lpszType

A pointer to the type of resource. For a list of resource types,