References:

“Introduction to MFC”, Deitel, Deitel, Nieto & Strassberger, Prentice Hall © 2000

http://vig.prenhall.com/catalog/academic/product/1,4096,0130132497,00.html

http://www.amazon.com/exec/obidos/ASIN/0130161470/deitelassociatin

“Programming Windows® with MFC”, Jeff Prosise, Microsoft Press © 1999

http://www.microsoft.com/mspress/books/1431.asp

http://www.amazon.com/exec/obidos/ASIN/1572316950/qid=1007995908/sr=1-1/ref=sr_1_15_1/103-8969452-5438253
MFC Class Hierarchy

Common window/GUI behaviors implemented in base classes for derived classes to inherit and over-ride for specific application behavior.

Only a small subset of the MFC class hierarchy will be covered herein.

**CWinApp**: a base class for creating & executing Win applications

**CDialog**: a base class for creating & managing dialog windows

**CFrameWnd**: a base class for creating & managing frame windows
GUI Events

GUI programs are event driven. Applications responds to user actions which generates events, (mouse, keyboard, etc.). Events are communicated to a program through messages sent by the Op Sys. Program processing characterized by an event-loop which receives and responds to messages. MFC provides the necessary event-loop logic in an object-oriented application framework.

Message Handling

A MFC application, (app) derives classes from the MFC hierarchy that contain code to react to events (message handlers). Every message, (msg), is specified by a unique int, (message identifier). Message handlers are associated with message identifiers through a message map. The map registers handlers with the event-loop. When the app receives a msg it looks up the identifier to find the corresponding handler to execute for the appropriate response. The map is defined by the DECLARE_MESSAGE_MAP() macro:

```
//Message Map
BEGIN_MESSAGE_MAP(owner-class, base-class)
   ON_COMMAND(message-id, msg-handler)
END_MESSAGE_MAP()
```

Predefined msg ids in header file: <afxwin.h>.
Resource Definitions

MSVC++ supports a resource definition language, (RDL), for the specification of GUI controls: (type, location, size, msg id, etc.).

RDL statements are stored in a resource.rc file in a Win32 application project.

Resource files can be created & edited by hand, but usually the IDE resource editor is used to graphically design the interface controls. A resource compiler translates resource.rc files into object code.
Variable Prefix Naming Convention

To more easily identify type and scope, (w/o having to refer back to the definition), most MFC programmers employ Hungarian notation

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ar</td>
<td>array</td>
</tr>
<tr>
<td>b</td>
<td>Boolean</td>
</tr>
<tr>
<td>c</td>
<td>Char</td>
</tr>
<tr>
<td>C</td>
<td>Class</td>
</tr>
<tr>
<td>dw</td>
<td>DWORD, double word or unsigned long</td>
</tr>
<tr>
<td>fn</td>
<td>Function</td>
</tr>
<tr>
<td>h</td>
<td>Handle</td>
</tr>
<tr>
<td>i</td>
<td>int (integer)</td>
</tr>
<tr>
<td>m</td>
<td>member</td>
</tr>
<tr>
<td>n</td>
<td>short int</td>
</tr>
<tr>
<td>p</td>
<td>a pointer variable containing the address of a variable</td>
</tr>
<tr>
<td>s</td>
<td>string</td>
</tr>
<tr>
<td>sz</td>
<td>ASCIIZ null-terminated string</td>
</tr>
<tr>
<td>s_</td>
<td>static class member variable</td>
</tr>
<tr>
<td>w</td>
<td>WORD unsigned int</td>
</tr>
</tbody>
</table>
Win32 Application Project
In MSVC++ create an empty Win32 Application project, (not a console application).

MFC library
Select menu/option: Project / Settings...
Select “Use MFC in a Shared DLL.

Linking to the MFC DLL decreases exe size & compilation time.
HelloWorld.h

```cpp
1. //HelloWorld.h
2. class CHelloWindow : public CFrameWnd {
3. public:
4.   CHelloWindow(); // constructor initializes window
5.   ~CHelloWindow(); // destructor releases resources

6. private:
7.   CStatic m_Hello; // contains Hello World string
8.};
```

Line 4. derives from CFrameWnd inheriting basic window functionality.

Line 7. Defines a MFC CStatic object.

![Screen coordinate system diagram](image)

**screen coordinate system**
9. //HelloWorld.cpp
10. // include application framework windows class definitions:
11. #include <afxwin.h> // application frameworks header
12. #include "HelloWorld.h" // class definition for application

13. // constructor initializes the window
14. CHelloWindow::CHelloWindow()
15. {
16.   // Create Window with Title Bar
17.   Create( NULL, // default CFrameWnd class
18.             "Hello", // window title
19.             WS_OVERLAPPEDWINDOW, // full-featured window
20.             CRect( 200, 100, 350, 200 ) ); // screen coordinates

21.   m_Hello.Create( // create Windows control
22.                 "Hello World", // text
23.                 WS_CHILD | WS_VISIBLE | WS_BORDER // window styles
24.                 | SS_CENTER, // static object styles
25.                 CRect( 20, 30, 120, 50 ), // window coordinates
26.                 this ); // context that owns child window
27. }
29. CHelloWindow::~CHelloWindow()
30. {
31. }

32. // derive application class from CWinApp
33. class CHelloApp : public CWinApp {
34. public:
35.     BOOL InitInstance()  // override default function
36.     {
37.         m_pMainWnd = new CHelloWindow();    // create window
38.         m_pMainWnd->ShowWindow( m_nCmdShow );  // make visible
39.         m_pMainWnd->UpdateWindow();    // force refresh
40.         return TRUE;                  // report success
41.     }
42. } HelloApp;  // instantiate application
Window Creation

Line 11. (\texttt{#include <afxwin.h> // application frameworks header})

Includes standard MFC message Ids, handlers and map.

Lines 17-20:

\begin{verbatim}
17. Create( NULL, // default CFrameWnd class
18. "Hello", // window title
19. WS_OVERLAPPEDWINDOW, // full-featured window
20. CRect( 200, 100, 350, 200 ) ); // screen coordinates
\end{verbatim}

Creates the main application window. The \texttt{NULL} argument instructs Windows to use the default window properties for this window class. The \texttt{WS_OVERLAPPEDWINDOW} setting creates a resizable window with standard window controls. The last \texttt{Create()} argument instantiates a \texttt{CRect(rectangle)} object to store the window screen coordinates. The first (x,y) pair gives the top-left coordinate and the last (x,y) pair gives the lower-right coordinate. This defines a window that is 150 x 100 pixels, (width, height).
Hello World Explanation (continued)

CStatic creation

Lines 21-27: creates a CStatic object (used for text labels).

```cpp
21. m_Hello.Create( // create Windows control
22. "Hello World",    // text
23. WS_CHILD | WS_VISIBLE | WS_BORDER // window styles
24. | SS_CENTER,        // static object styles
25. CRect( 20, 30, 120, 50 ), // window coordinates
26. this );         // context that owns child window
27. }
```

The first argument is the text label to be displayed. The second argument is mask to set the CStatic window characteristics. It is formed by logically OR’ing together pre-defined window style, (WS), constants: (WS_CHILD : sub-window; WS_VISIBLE : viewable; WS_BORDER : rectangular border; SS_CENTER : center text).

Line 26: gives the owner (parent) argument for the CStatic subwindow, this window is the CHelloWindow widow, (establishes an association between the sub-window and parent window, which allows the MS Win OS to move the window in memory).
Win Application Class: Lines 33-42:

```cpp
32. // derive application class from CWinApp
class CHelloApp : public CWinApp {
34. public:
35.   BOOL InitInstance() // override default function
36.   {
37.     m_pMainWnd = new CHelloWindow(); // create window
38.     m_pMainWnd->ShowWindow( m_nCmdShow ); // make visible
39.     m_pMainWnd->UpdateWindow(); // force refresh
40.     return TRUE; // report success
41.   }
42. } HelloApp; // instantiate application
```

All MFC app’s must have 1 (& only 1) instance of a class derived from CWinApp. The CWinApp class controls application instantiation, execution (event loop), and destruction. The main() is replaced is replaced by CWinApp.InitInstance() instantiates the app main window object and begins execution. The <x>Window() FNs above are inherited from CWinApp. MFC apps (must) use a pre-defined BOOL (int) type with TRUE/FALSE constants instead of the standard C++ bool type. The m_nCmdShow inherited variable indicates that the win is to be initially displayed unmaximized and unminimized.
A trivial menu options counter application.

The following MFC code displays a window with a few menus, allowing a user to select options from Menu1 and Menu2. The Show menu count option then displays a count of the number of options selected from the preceding two menus in a message dialog box.
Code

1. // WinMenus.h
2. // create menus with MFC
3.
4. class CMenusWin : public CFrameWnd {
5. public:
6.   CMenusWin();
7.   afx_msg void OnExit();
8.   afx_msg void OnCount();
9.   afx_msg void OnShowCount();
10. private:
11.   int m_iTotal;    // count menu options selected
12.   ostrstream m_str;  // output string stream
13.   DECLARE_MESSAGE_MAP();
14. };
15. // WinMenus.cpp
16. // create simple menus with MFC
17. #include <afxwin.h>  // MFC application framework
18. #include <strstrea.h> // C-style string stream class
19. #include <iomanip.h> // I/O manipulators
20. #include "WinMenusIDs.h" // application message ID symbols
21. #include "WinMenus.h"

22. CMenusWin::CMenusWin() { // construct window
23. {
24.     Create( NULL, "Menus Example", WS_OVERLAPPEDWINDOW,
25.             CRect( 100, 100, 300, 300 ), NULL, "Count" );
26.     m_iTotal = 0;
27. }

- The CRect constructor could have been replaced by the MFC pre-defined CRect object rectDefault, to allow Windows to choose the initial size and placement.

- The second NULL argument indicates that this is a root window having no parent.

- The "Count" argument associates the menu defined in the resource file with the window.
28. // afx_msg precedes each message handler function
29. afx_msg void CMenusWin::OnExit()
30. {
31. SendMessage( WM_CLOSE );
32. }
33. // count each menu option selected
34. void CMenusWin::OnCount()
35. {
36. m_iTotal++;
37. }
38. afx_msg void CMenusWin::OnShowCount()
39. {
40. m_str.seekp( 0 ); // reset output string stream
41. m_str << setprecision( 2 )
42. << setiosflags( ios::fixed | ios::showpoint )
43. << "Options = " << m_iTotal << ends; // stopper
44. // display new dialog box with output string
45. MessageBox(m_str.str(), "Options:" );
46. }

afx_msg is the MFC prefix used to mark a msg handler. The WM_CLOSE msg terminates execution.

Msg handler FN for all Menu1 & Menu2 options, to update the option selection counter. It receives & handles a range of msg Ids.

Returns a C-style string (char *) from the ostrstream m_str object.

MessageBox FN displays a popup msg dialog window. It accepts a C-style string to display and a dialog win label string.
47. BEGIN_MESSAGE_MAP( CMenusWin, CFrameWnd )
48. ON_COMMAND( IDM_EXIT, OnExit )
49. ON_COMMAND_RANGE(IDM_M101, IDM_M203, OnCount)
50. ON_COMMAND( IDM_SHOW_COUNT, OnShowCount )
51. END_MESSAGE_MAP()

52. class CMenusApp : public CWinApp {
53. public:
54. BOOL InitInstance() // called by CWinApp::CWinApp
55. {
56.    m_pMainWnd = new CMenusWin; // create window
57.    m_pMainWnd->ShowWindow( m_nCmdShow ); // make it visible
58.    m_pMainWnd->UpdateWindow(); // force refresh
59.    return TRUE; // report success
60. }
61. }

62. } menusApp; // calls CWinApp::CWinApp

Invokes msg map macro to associate msg IDs with handler FNs.

IDM_ prefix indicates an identifier of a menu using MFC naming conventions. Msg handlers have the prefix On.

Derives the app class from CWinApp and instantiates it.
Message Identifiers

Predefined MFC message identifiers are in the range: [0 … 1023]
Programmer-defined message identifiers are in the range: [1024 … 65535]

63. // WinMenusIDs.h
64. // define messages used by menus.cpp and menus.rc

65. #define IDM_EXIT 2000  // File/Exit msg id
66. #define IDM_M1O1 2011  // Menu1 msg ids
67. #define IDM_M1O2 2012
68. #define IDM_M2O1 2021  // Menu2 msg ids
69. #define IDM_M2O2 2022
70. #define IDM_M2O3 2023
71. #define IDM_SHOW_COUNT 2031  // Show/Count msg id

Message Ids support the connections between the messages and associated handlers.
WinMenus.rc resource file defines the menu and options to msg Id associations. The lines are resource definition statements, (the MS Win GUI description language). Can be created in a text editor and added to the project.

Note: creating a resource file within the MS VC IDE and opening it will invoke the graphical resource editor which is not covered in these notes.
Dialog Based Application

The code on the following slides discusses the code for simple MFC dialog window based application. The program allows a user to enter a date for the Gregorian calendar and displays the corresponding day of the week for the date.

The code introduces dialog boxes, button controls and edit text/box controls.

Error checking for valid Gregorian calendar dates is not included to focus on the MFC code. The application will currently accept invalid dates and incorrectly formatted user input.
// DayWeek_ids.h
// Define Message Numbers
#define IDC_DATE 2000
#define IDC_DAY 2001
#define IDC_WEEK 2002
#define IDC_CLEAR 2003

1. // DayWeek.h
2. // Day of Week MFC dialog program
3. class CDayWeek : public CDialog {
4. public:
5.   CDayWeek()
6.   : CDialog( "DayWeek" ), m_nDay(1), m_nMon(1), m_nYear(2000)
7. {
8.    
9.  private:
10.    int m_nDay, m_nMon, m_nYear; // Date
11.    DECLARE_MESSAGE_MAP()
12. };
13. // DayWeek.cpp
14. // Day of Week MFC dialog program
15. #include <afxwin.h>
16. #include <strstrea.h> // C-string streams
17. #include <string>
18. #include "DayWeek_ids.h"
19. #include "DayWeek.h"

20. // clicked the "Day" button
21. afx_msg void CDayWeek::OnDay()
22. {
23.    const int TEXT_SIZE = 11;
24.    char tmp, szText[TEXT_SIZE + 1]; // buffer for conversions
25.    // get addresses of Edit Box Controls
26.    CEdit *pDate = (CEdit *) (GetDlgItem(IDC_DATE));
27.    CEdit *pWeek = (CEdit *) (GetDlgItem(IDC_WEEK));
28.    pDate->GetWindowText(szText, TEXT_SIZE); // get Date
29.    std::string DayOfWeek(szText); // initialize string

Lines 26-27: get the addresses of the edit dialog boxes for manipulation. The IDC_ codes, defined in DayWeek_ids.h, are passed to GetDlgItem() for the address lookup.

GetDlgItem() returns base type (CWnd*) pointers, which are type cast to the derived pointer type, (CEdit*).

GetWindowText() returns the dialog edit box control text, (C-style string), using the dialog control pointer.
DayWeek.cpp (continued)

Code to parse input and compute day of week.

30. if (DayOfWeek[0] == '0') DayOfWeek.erase(0,1); //del leading day zero
31. int m1 = DayOfWeek.find("/",0) + 1; //find start of month char position
32. if (DayOfWeek[m1] == '0') DayOfWeek.erase(m1,1); //del leading mon zero

33. istringstream istr((char*) DayOfWeek.c_str()); //init input istringstream
34. istr >> m_nMon >> tmp >> m_nDay >> tmp >> m_nYear; //read date

35. if (m_nMon < 3) { //formula requires Jan & Feb
36.   m_nMon += 12; //be computed as the 13th & 14th months
37.   m_nYear -= 1; //of the preceding year
38. } //if

39. switch ( (m_nDay + 2 * m_nMon + 3 * (m_nMon + 1) / 5 + m_nYear
40.          + m_nYear / 4 - m_nYear / 100 + m_nYear / 400 + 1) % 7 ) {
41.   case 0: DayOfWeek = "Sunday";      break;
42.   case 1: DayOfWeek = "Monday";      break;
43.   case 2: DayOfWeek = "Tuesday";     break;
44.   case 3: DayOfWeek = "Wednesday";   break;
45.   case 4: DayOfWeek = "Thursday";    break;
46.   case 5: DayOfWeek = "Friday";      break;
47.   case 6: DayOfWeek = "Saturday";    break;
48.   default: DayOfWeek = "Error";     break;
49. } //switch
50. pWeek->SetWindowText( DayOfWeek.c_str() ); // display week day
51. pDate->SetFocus(); // next date
52. } // OnDay()

50. // clicked the "Clear" button
51. afx_msg void CDayWeek::OnClear()
52. {
53. // get addresses of Edit Box Controls
54. CEdit *pDate = ( CEdit * ) ( GetDlgItem( IDC_DATE ) ) ;
55. CEdit *pWeek = ( CEdit * ) ( GetDlgItem( IDC_WEEK ) ) ;
56. m_nDay = m_nMon = 1; m_nYear = 2000;
57. pDate->SetWindowText( "" ); // clear the date edit box
58. pWeek->SetWindowText( "" ); // clear the week day box
59. pDate->SetFocus(); // next date to input
60. } // OnClear()

The GetDlgItem() FN must be re-called every time to manipulate the controls, (because the OS reallocates memory every time Windows are created and destroyed).
BEGIN_MESSAGE_MAP( CDayWeek, CDialog )
on_command( IDC_DAY, OnDay )
on_command( IDC_CLEAR, OnClear )
END_MESSAGE_MAP()

// dialog-based application
class CDayWeekApp : public CWinApp {
public:
  BOOL InitInstance()
  {
    CDayWeek DayWeekDialog;
    DayWeekDialog.DoModal(); // run dialog
    return FALSE; // finished
  }
}

Derives the app class from CWinApp and instantiates it. The dialog window is instantiated and the DoModal() is invoked to display it as modal window, (modal windows require the user to respond to them before anything else can be done).
// DayWeek.rc
// resource script for DayWeek
#include "afxres.h"
#include "DayWeek_ids.h"

DayWeek DIALOG 50, 50, 130, 130
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU

CAPTION "Day of Week"
{
LTEXT "Enter Date (MM/DD/YYYY):", IDC_STATIC, 30, 20, 98, 8
EDITTEXT IDC_DATE, 30, 30, 46, 16, ES_AUTOHSCROLL

DEFPUSHBUTTON "Day", IDC_DAY, 50, 50, 30, 15

LTEXT "Week Day:", IDC_STATIC, 30, 70, 50, 8
EDITTEXT IDC_WEEK, 30, 80, 42, 16,
ES_READONLY | NOT WS_TABSTOP

PUSHBUTTON "Clear", IDC_CLEAR, 50, 100, 30, 15,
NOT WS_TABSTOP
}

IDC_STATIC controls do not generate msgs, (does not require control IDC_number). LTEXT is a left aligned control, (RTEXT, CTEXT also available). WS_SYSMENU includes a system win menu with Move & Close options (right-click title bar). WS_POPUP wins are parentless.

Dialog win location: upper left corner is at (50,50) pixels. The last 2 numbers give horizontal, vertical dialog unit size. Horizontal units = 0.25 char width, vertical = 0.125 char height.

Dialog box title.

Dialog styles prefixed with DS_.

Edit style allows horizontal scrolling w/o scroll bar.

Defines a output control w/o focus, (when user hits tab key control is skipped).