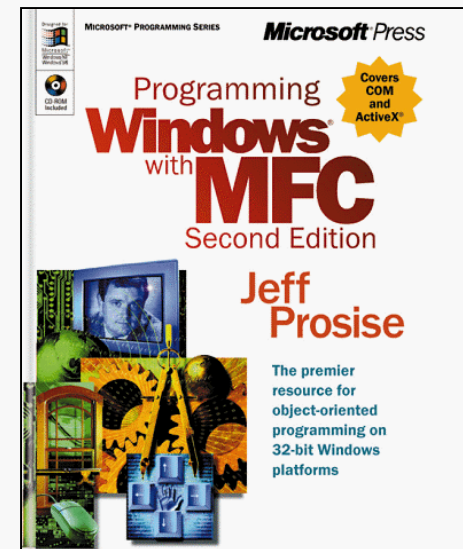
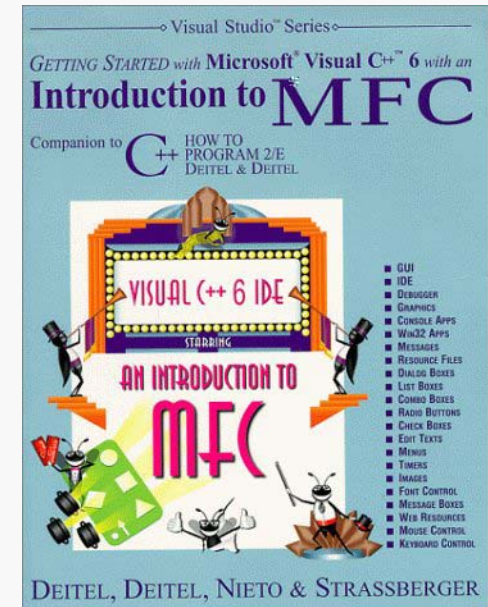


References:

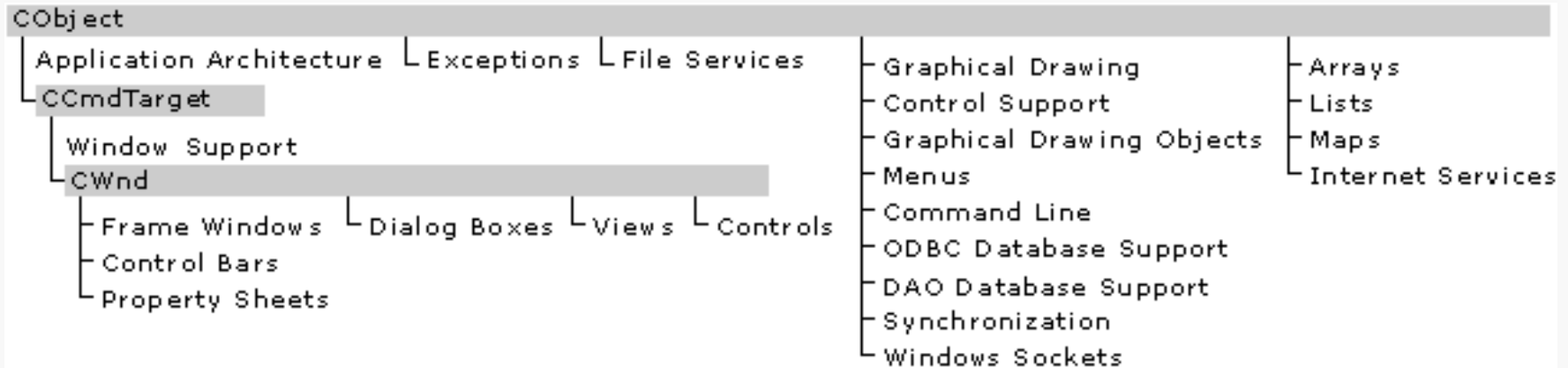
“Introduction to MFC”, Deitel, Deitel, Nieto & Strassberger, Prentice Hall © 2000

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



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

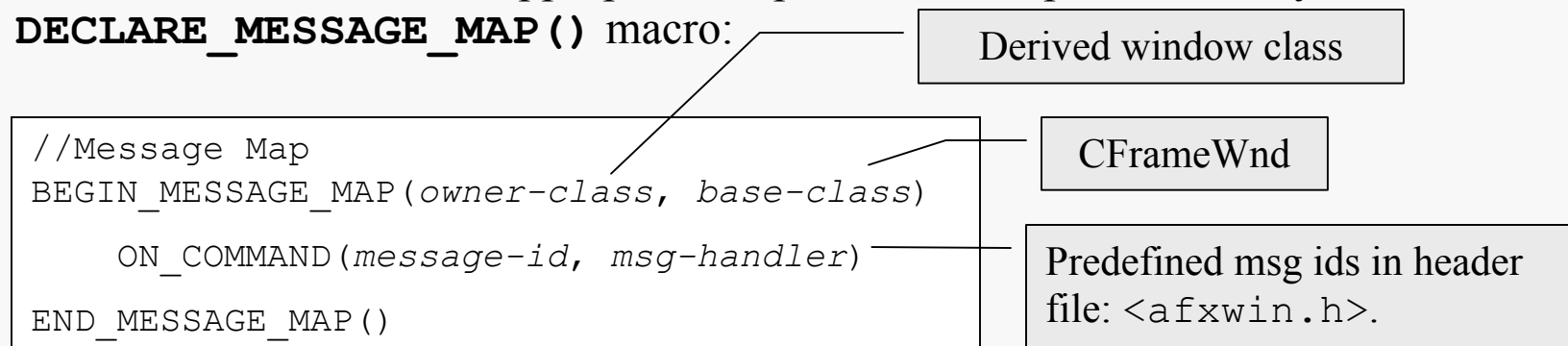
CFrameWnd: a base class for creating & managing frame windows

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:

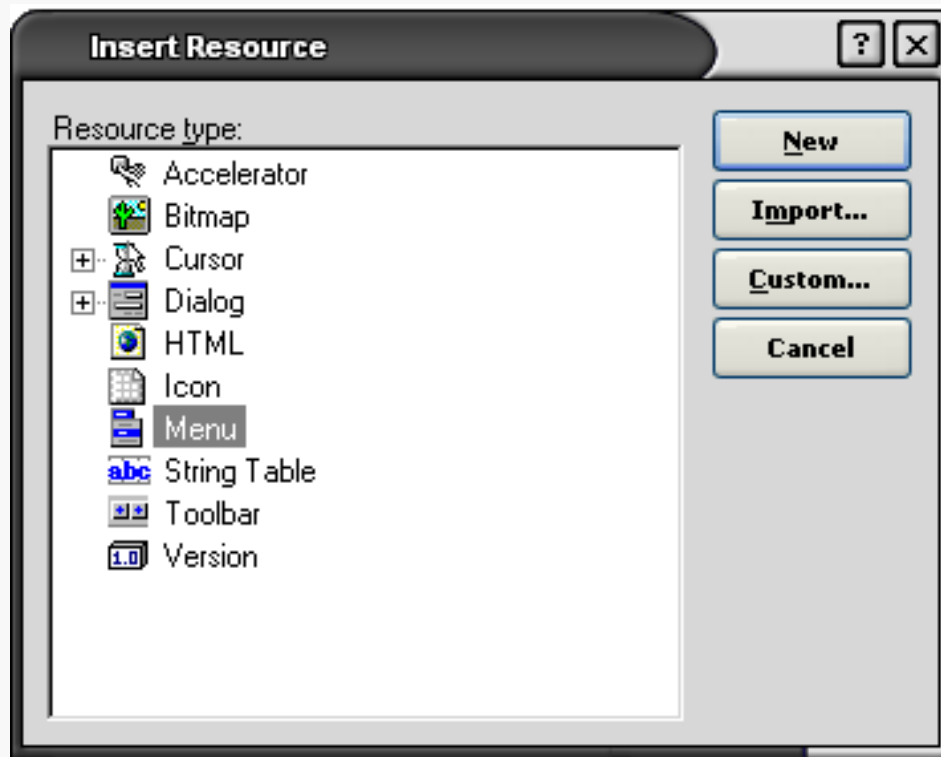


Resource Definitions

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

RDL statements 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 (Charles Simonyi) notation:

Prefix	Data type
ar	array
b	Boolean
c	Char
C	Class
dw	DWORD, double word or unsigned long
fn	Function
h	Handle
i	int (integer)
m	member
n	short int
p	a pointer variable containing the address of a variable
s	string
sz	ASCIIZ null-terminated string
s_	static class member variable
w	WORD unsigned int

Win32 Application Project

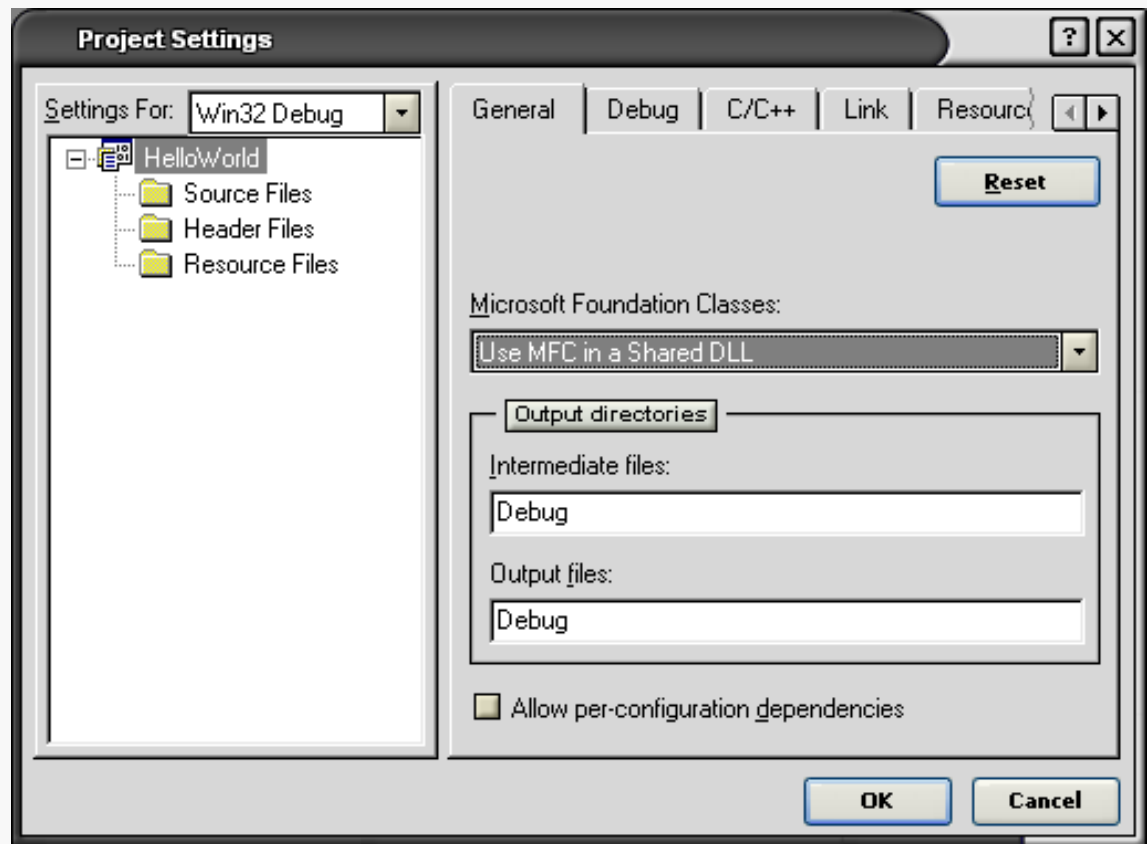
In MSVC++ create an empty Win32 Application project, (not a console application).

MFC library

Select menu/option: Project / Settings...

Select "Use MFC in a Shared DLL.

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

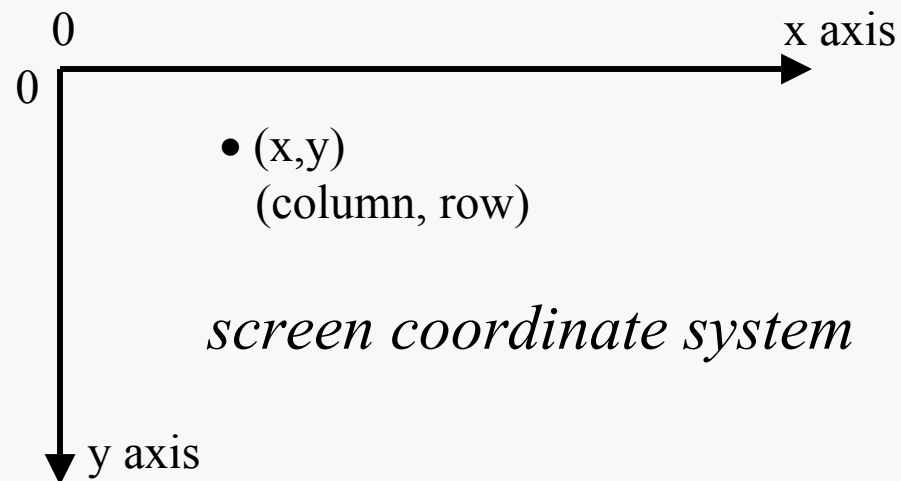


Code

```
1. //HelloWorld.h
2. class CHelloWindow : public CFrameWnd {
3. public:
4.     CHelloWindow(); // constructor initializes window
5.     ~CHelloWindow(); // destructor releases resources
6. private:
7.     CString m_Hello; // contains Hello World string
8. };
```

Line 4. derives from CFrameWnd inheriting basic window functionality.

Line 7. Defines a MFC CString object.



```
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. (`#include <afxwin.h>` // application frameworks header)

Includes standard MFC message Ids, handlers and map.

Lines 17-20:

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

Creates the main application window. The `NULL` argument instructs Windows to use the default window properties for this window class. The `WS_OVERLAPPEDWINDOW` setting creates a resizable window with standard window controls. The last `Create()` argument instantiates a `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).

CStatic creation

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

```
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` window, (establishes an association between the sub-window and parent window, which allows the MS Win OS to move the window in memory).

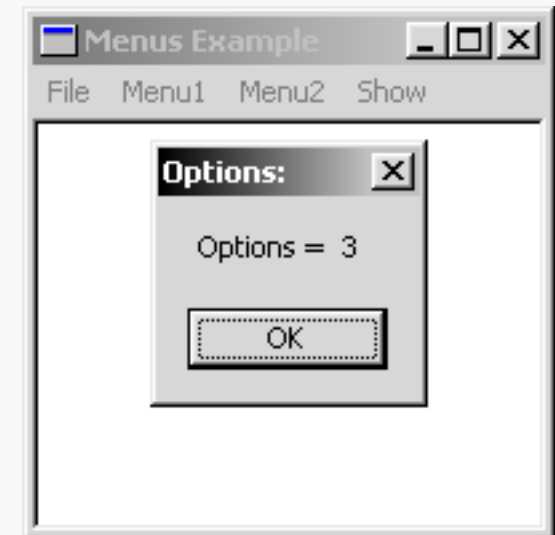
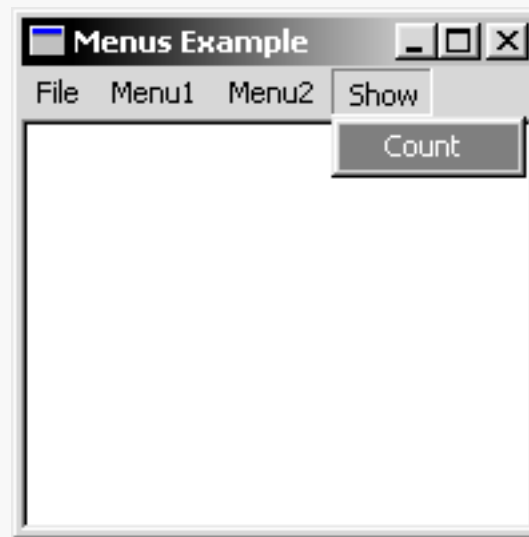
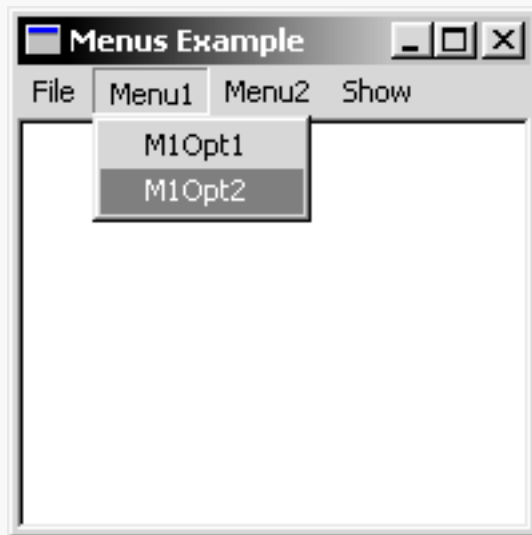
Win Application Class: Lines 33-42:

```
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
```

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 CMenuWin : public CFrameWnd {
5. public:
6.     CMenuWin();
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.     ostream m_str;      // output string stream
13.     DECLARE_MESSAGE_MAP()
14. };
```

Menu “callback” FNs.
Invoked when menu options
are selected.

Invokes the message map
macro to map message IDs
to the message handler FNs.

```
15. // WinMenus.cpp
16. // create simple menus with MFC
17. #include <afxwin.h>           // MFC application framework
18. #include <strstream.h>       // C-style string stream
19. #include <iomanip.h>         // I/O manipulators
20. #include "WinMenusIDs.h"    // application message ID symbols
21. #include "WinMenus.h"

22. CMenuWin::CMenuWin()        // 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 CMenuWin::OnExit() ←
30. {
31.     SendMessage( WM_CLOSE );
32. }

33. // count each menu option selected
34. void CMenuWin::OnCount( ) ←
35. {
36.     m_iTotal++;
37. }

38. afx_msg void CMenuWin::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.

Converts ostream m_str into a C-style string.

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( CMenuWin, CFrameWnd )
48.     ←
49.     ON_COMMAND( IDM_EXIT, OnExit )
50.     ON_COMMAND_RANGE( IDM_M101, IDM_M203, OnCount )
51.     ON_COMMAND( IDM_SHOW_COUNT, OnShowCount )
52. END_MESSAGE_MAP()

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

62. } menuApp;                               // 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.

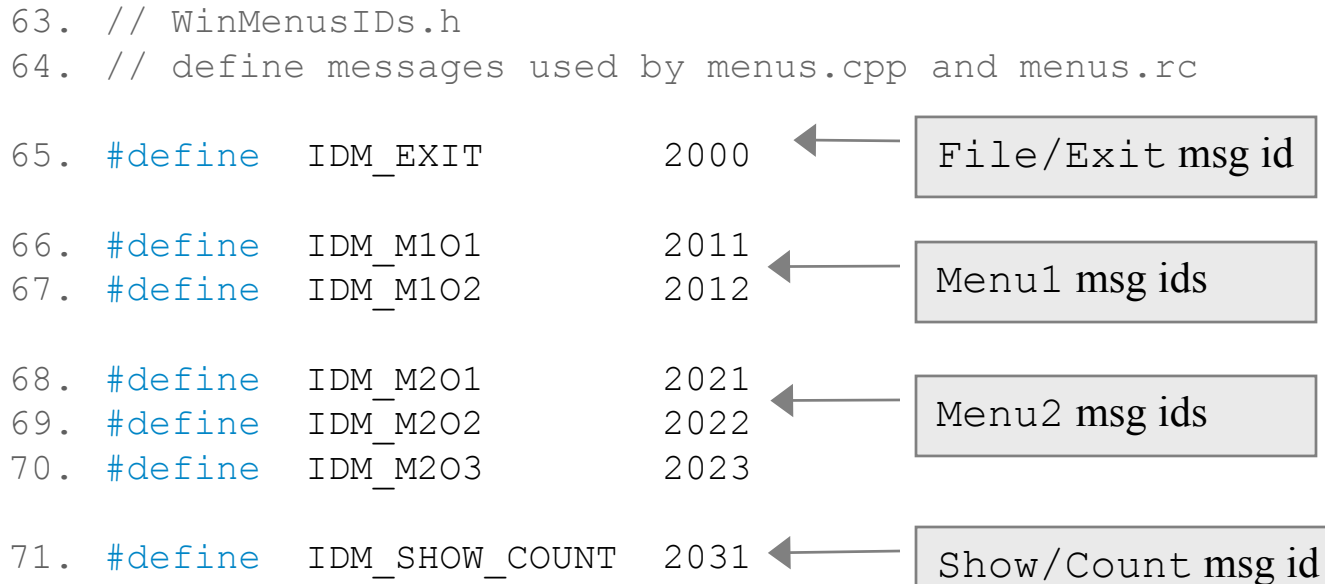
Message Identifiers

Predefined MFC message identifiers are in the range: [0 ... 1023]

Programmer-defined message identifiers are in the range: [1024 ... 65535]

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

65. #define   IDM_EXIT           2000 ← File/Exit msg id
66. #define   IDM_M101          2011 ← Menu1 msg ids
67. #define   IDM_M102          2012
68. #define   IDM_M201          2021 ← Menu2 msg ids
69. #define   IDM_M202          2022
70. #define   IDM_M203          2023
71. #define   IDM_SHOW_COUNT    2031 ← Show/Count msg id
```

The diagram shows a list of C++ preprocessor directives for defining message identifiers. Each line is numbered from 63 to 71. Lines 65, 66, 68, and 71 have arrows pointing from the numeric value to a grey rectangular box containing a descriptive label. The labels are: 'File/Exit msg id' for 2000, 'Menu1 msg ids' for 2011, 'Menu2 msg ids' for 2021, and 'Show/Count msg id' for 2031. Lines 67, 69, and 70 do not have arrows or labels.

Message Ids support the connections between the messages and associated handlers.

```
72. // WinMenus.rc
73. // resource script for menus example
74. #include <afxres.h>
75. #include "WinMenusIDs.h"

76. Count MENU
77. {
78.     POPUP "File"
79.     {
80.         MENUITEM "Exit", IDM_EXIT
81.     }

82.     POPUP "Menu1"
83.     {
84.         MENUITEM "M1Opt1", IDM_M1O1
85.         MENUITEM "M1Opt2", IDM_M1O2
86.     }

87.     POPUP "Menu2"
88.     {
89.         MENUITEM "M2Opt1", IDM_M2O1
90.         MENUITEM "M2Opt2", IDM_M2O2
91.         MENUITEM "M2Opt3", IDM_M2O3
92.     }

93.     POPUP "Show"
94.     {
95.         MENUITEM "Count", IDM_SHOW_COUNT
96.     }
97. }
```

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.

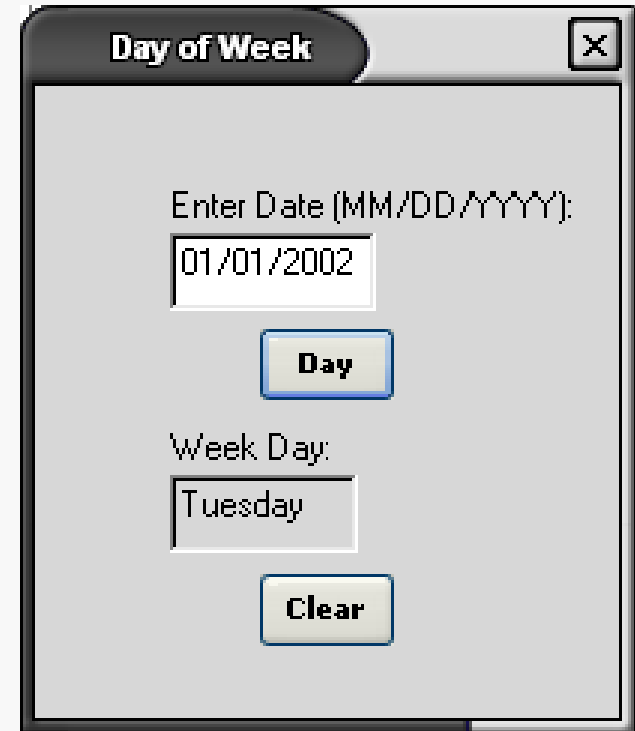
Note: creating a resource file within the MS VC IDE and opening it will invoke the graphical resource editor which is not covered in these notes.

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
```

Defines the dialog controls message Ids.

```
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.     afx_msg void OnDay();           // clicked the "Day" button
10.    afx_msg void OnClear();         // clicked the "Clear" button
11.
12. private:
13.     int m_nDay, m_nMon, m_nYear;   // Date
14.
15.     DECLARE_MESSAGE_MAP()
16. };
```

Dialog resource "DayWeek" is passed to inherited Cdialog constructor to pass dialog attributes.

Dialog box button "callback" FNs. Invoked when buttons are clicked.

```
1. // DayWeek.cpp
2. // Day of Week MFC dialog program
3. #include <afxwin.h>
4. #include <string>
5. #include <sstream>
6. #include "DayWeek_ids.h"
7. #include "DayWeek.h"

8. // clicked the "Day" button
9. afx_msg void CDayWeek::OnDay()
10. {
11.     const int TEXT_SIZE = 11;
12.     char tmp, szText[ TEXT_SIZE * 1 ]; // buffer for conversions
13.
14.     // get addresses of Edit Box Controls
15.     CEdit *pDate = ( CEdit * ) ( GetDlgItem( IDC_DATE ) );
16.     CEdit *pWeek = ( CEdit * ) ( GetDlgItem( IDC_WEEK ) );
17.
18.     pDate->GetWindowText( szText, TEXT_SIZE ); // get Date text
19.     std::string DayOfWeek(szText);           // init string
```

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.

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.

Code to parse input and compute day of week.

```
1.  std::istringstream istr( DayOfWeek );          // init input sstream
2.  istr >> m_nMon >> tmp >> m_nDay >> tmp >> m_nYear; //read date
3.
4.  if (m_nMon < 3) {          // formula requires Jan & Feb
5.      m_nMon += 12;          // be computed as the 13th & 14th months
6.      m_nYear -= 1;          // of the preceding year
7.  } //if
8.
9.  switch ( (m_nDay + 2 * m_nMon + 3 * (m_nMon + 1) / 5 + m_nYear
10.         + m_nYear / 4 - m_nYear / 100 + m_nYear / 400 + 1) % 7) {
11.  case 0: DayOfWeek = "Sunday";    break;
12.  case 1: DayOfWeek = "Monday";    break;
13.  case 2: DayOfWeek = "Tuesday";   break;
14.  case 3: DayOfWeek = "Wednesday"; break;
15.  case 4: DayOfWeek = "Thursday";  break;
16.  case 5: DayOfWeek = "Friday";    break;
17.  case 6: DayOfWeek = "Saturday";  break;
18.  default: DayOfWeek = "Error";
19.  } //switch
```

```
50.  pWeek->SetWindowText( DayOfWeek.c_str() ); // display week day
51.    pDate->SetFocus(); // next date
52. } // OnDay()
```

```
53. // clicked the "Clear" button
54. afx_msg void CDayWeek::OnClear() {
```

```
55.     // get addresses of Edit Box Controls
56.     CEdit *pDate = ( CEdit * ) ( GetDlgItem( IDC_DATE ) );
57.     CEdit *pWeek = ( CEdit * ) ( GetDlgItem( IDC_WEEK ) );
58.
59.     m_nDay = m_nMon = 1; m_nYear = 2000;
60.
61.     pDate->SetWindowText( "" ); // clear the date edit box
62.     pWeek->SetWindowText( "" ); // clear the week day box
63.     pDate->SetFocus(); // next date to input
64. } // OnClear()
```

SetWindowText() sets the contents of the dialog text box, using the dialog control pointer. SetFocus() makes the control *active*, (ie, user doesn't need to click it to edit the date text).

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).

```
64. BEGIN_MESSAGE_MAP( CDayWeek, CDialog )
65.     ON_COMMAND( IDC_DAY, OnDay )
66.     ON_COMMAND( IDC_CLEAR, OnClear )
67. END_MESSAGE_MAP()

68. // dialog-based application
69. class CDayWeekApp : public CWinApp {
70. public:
71.     BOOL InitInstance()
72.     {
73.         CDayWeek DayWeekDialog;
74.         DayWeekDialog.DoModal(); // run dialog
75.         return FALSE;           // finished
76.     }
77.
78. } DayWeek;
```

The message macro maps the dialog button message IDs to the button handler FNs.

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).

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).

```

78. // DayWeek.rc
79. // resource script for DayWeek
80. #include "afxres.h"
81. #include "DayWeek_ids.h"

82. DayWeek DIALOG 50, 50, 130, 130
83. STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU

84. CAPTION "Day of Week"
85. {
86.     LTEXT      "Enter Date (MM/DD/YYYY):", IDC_STATIC, 30, 20, 98, 8
87.     EDITTEXT   IDC_DATE, 30, 30, 46, 16, ES_AUTOHSCROLL
88.     DEFPUSHBUTTON "Day", IDC_DAY, 50, 50, 30, 15
89.     LTEXT      "Week Day:", IDC_STATIC, 30, 70, 50, 8
90.     EDITTEXT   IDC_WEEK, 30, 80, 42, 16,
91.     ES_READONLY | NOT WS_TABSTOP
92.     PUSHBUTTON "Clear", IDC_CLEAR, 50, 100, 30, 15,
93.     NOT WS_TABSTOP
94. }

```

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).

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.