Introducing MFC

Programming Windows with MFC, Second Edition. Jeff Prosise

Hello, MFC

Short Years Ago … Windows Applications written in C:

- Knowing the ins and outs of new operating system
- Knowing hundreds of different API functions

Today … C++ has become the professional Windows programmer’s language of choice.

MFC is a class library that:

- Abstracts the Windows API
- Encapsulates the basic behavior of Windows

Window uses the Win32 API.

SDK – Software Development Kit
API – Application Programming Interface
MFC – Microsoft Foundation Classes

The Windows Programming Model

Traditional Programs: Procedural programming model.

- Programs execute top-to-bottom.
- The path varies depending on the input and conditions.

Windows programs: Event-driven programming model.

- Applications respond to *events*, by processing messages sent by the operating system.

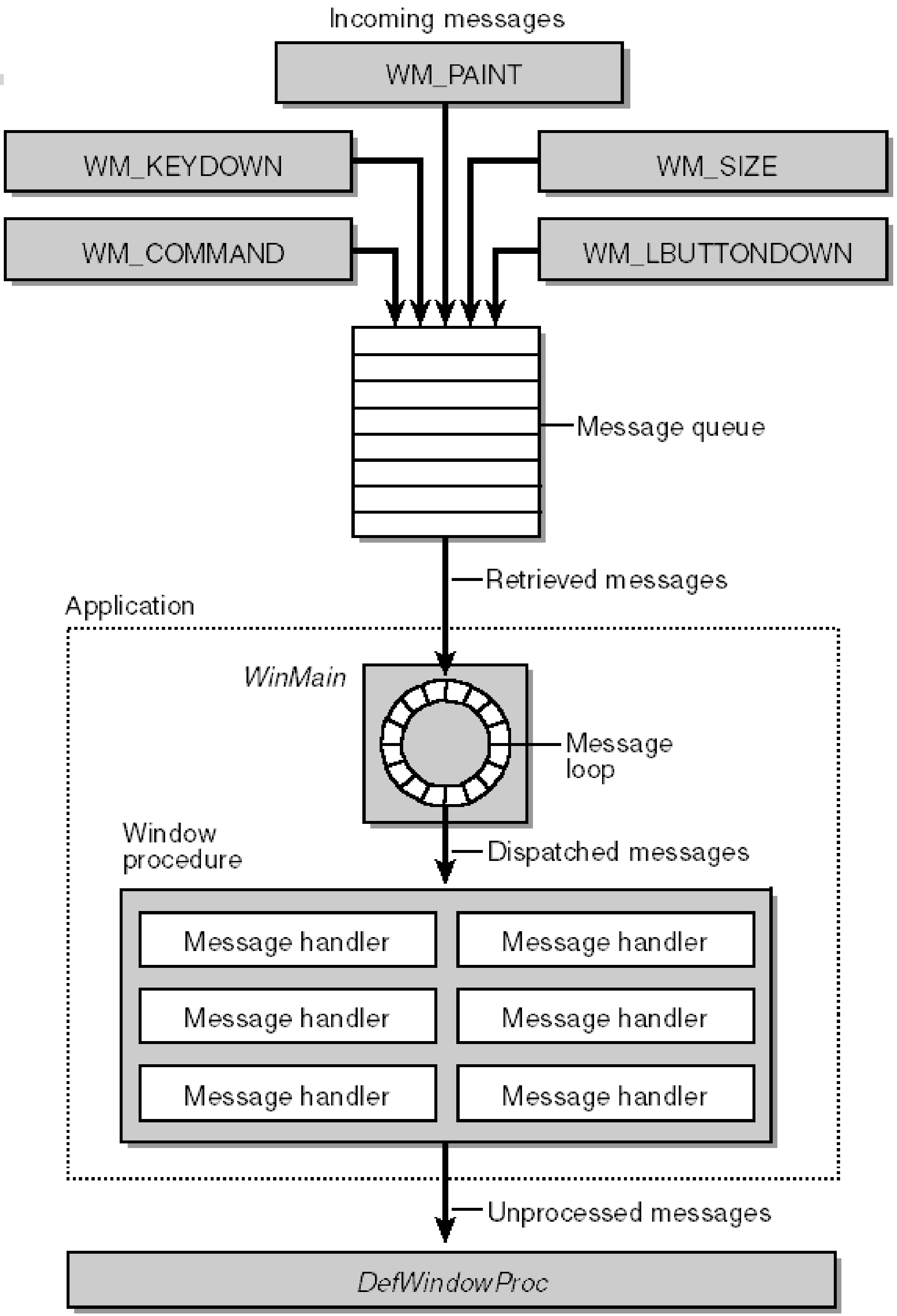
events – keystroke, mouse click, command for windows repaint …

The Message Handler can call local function, or API functions.

API functions such as creating a window, drawing a line, performing file I/O and many more.

API functions are contained in DLL’s.

DLL –
Dynamic Link Libraries



Messages, Messages

Message	***Sent When***
WM_CHAR	A character is input from the keyboard.
WM_COMMAND	The user selects an item from a menu, or a control sends a notification to its parent.
WM_CREATE	A window is created.
WM_DESTROY	A window is destroyed.
WM_LBUTTONDOWN	The left mouse button is pressed.
WM_LBUTTONUP	The left mouse button is released.
WM_MOUSEMOVE	The mouse pointer is moved.
WM_PAINT	A window needs repainting.
WM_QUIT	The application is about to terminate.
WM_SIZE	A window is resized.

Messages, Messages (Cont.)

The **MSG** structure contains message information from the message queue.

```
typedef struct tagMSG
{
      HWND hwnd;    // Uniquely identifies a window.
      UINT message;     // Specifies the msg type.
      WPARAM wParam;    // Additional information
      LPARAM lParam;    //          about the msg.
      …
} MSG;
```

For *wParam* and *lParam*, the exact meaning depends on the value of the **message** member. For WM_LBUTTONDOWN it is the state of the Ctrl or Shift keys, and the mouse coordinates.

Hungarian Notation

Prefix	***Data Type***
b	BOOL
c or *ch*	char
clr	COLORREF
cx, cy	Horizontal or vertical distance
dw	DWORD
h	Handle
l	LONG
n	int
p	Pointer
sz	Zero-terminated string
w	WORD
wnd	CWnd
str	CString
m_	**class member variable**

Note:

Prefixes can be combined:

pszName

m_nAge

Introducing MFC

- MFC is the C++ library Microsoft provides to place an object-oriented wrapper around the Windows API.
- No need to call the Windows API directly.
- Create objects from MFC classes and call member functions (some functions are thin wrappers around the API).

- Visual C++ Wizards.
- MFC is also an *application framework.*
- Helps define the structure of an application and handles many routing chores.
- Document / View architecture.

The First MFC Application

Hello.h

```
class CMyApp : public CWinApp {
public:
      virtual BOOL InitInstance ();
};

class CMainWindow : public CFrameWnd {
public:
      CMainWindow ();            // ctor


protected:
      afx_msg void OnPaint ();
      DECLARE_MESSAGE_MAP ()
};
```

Hello.cpp

```cpp
#include <afxwin.h>
#include "Hello.h"

CMyApp myApp;   // The one and only application

// CMyApp member functions
BOOL CMyApp::InitInstance () {
   m_pMainWnd = new CMainWindow; // create app window
   m_pMainWnd->ShowWindow (m_nCmdShow);
   m_pMainWnd->UpdateWindow ();  // force repaint
   return TRUE;     // otherwise application shutdown
}

// CMainWindow message map and member functions
BEGIN_MESSAGE_MAP (CMainWindow, CFrameWnd)
   ON_WM_PAINT ()
END_MESSAGE_MAP ()
```

Hello.cpp (cont.)

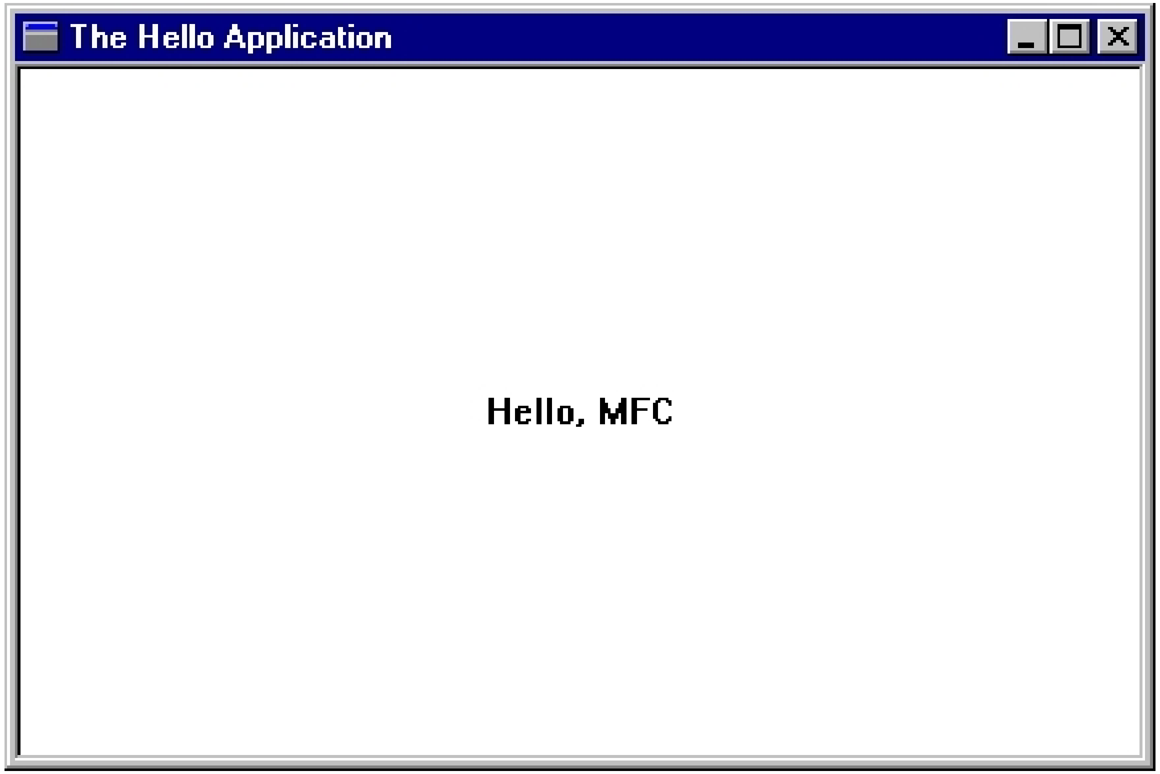
```
CMainWindow::CMainWindow () {
      Create (NULL, _T ("The Hello Application"));
}

void CMainWindow::OnPaint () {
      CPaintDC dc (this);           // Device context

      CRect rect;                   // Client rectangle
      GetClientRect (&rect);        // area

      dc.DrawText (_T ("Hello, MFC"), -1, &rect,
            DT_SINGLELINE | DT_CENTER | DT_VCENTER);
}
```

The Hello Application



Hello, MFC

The Application Object

The heart of an MFC application is an application object based on the *CWinApp* class.

CWinApp class

- provides the message loop that retrieves messages.
- dispatches them to the application’s window.

An application can have one, and only one, application object.

CMyApp declares no data members and overrides one inherited function.

- *InitInstance* is called early.
- *InitInstance* is where a window is created.

The *InitInstance* Function

CWinAp::InitInsance is a virtual function.

Its purpose is to provide the application with the opportunity to initialize itself.

At the very least, this means creating the window that will represent the application on the screen.

`m_pMainWnd` is a *CWinApp* object data member variable.

The Frame Window Object

Hello’s window class, *CMainWindow*, is derived from MFC’s *CFrameWnd* class.

A **Frame window** is a top level window that serves as an application’s primary interface to the outside world.

The window seen on the screen is created in *CMainWindow*’s constructor.

Create’s first argument, `NULL`, creates the default frame window.

Painting the Window

Hello’s window class get painted on WM_PAINT messages.

In Hello, these are processed by *OnPaint()*.

All graphical output is preformed through **device context** objects that abstract the physical destinations for output.

Default font and text color are used.

CPaintDC is a special case that is used **only** in WM_PAINT.

◀

The Message Map

How did a WM_PAINT message turn to a call to *OnPaint()* ?

A message map is a table that correlates messages and member functions. Some complex macros by MFC.

How to create a message map:

1. Declare it by adding DECLARE_MESSAGE_MAP.
2. Implement it by placing macros identifying messages between BEGIN_MESSAGE MAP and END_MESSAGE_MAP.
3. Add member function to handle the messages.

Any class derived from *CCmdTarget* can contain a message map.

Message Map (Cont.)

Usage of other generic macros:

```
   ON_MESSAGE(WM_SETTEXT, OnSetText)
```

With the declaration:

```
   afx_msg LRESULT OnSetText(WPARAM wParam,
                             LPARAM lParam);
```

Or command macros provided by

```
   ON_COMMAND(ID_FILE_OPEN, OnFileOpen)
```

For the **File → Open** command.

Drawing in a Window

The part of Windows responsible for graphics output is the GDI.

Windows created a Device Independent output model, allowing identical code for all graphics adapters, printers and other devices.

The **Device Context** object:

- Serves as the key that unlocks the door to output devices.
- Encapsulates the GDI functions used for output.

```
CPaintDC* pDC = new CPaintDC(this);
// Do some drawing

delete pDC;
```

◁

GDI – Graphics Device Interface

Device Context Classes

Class Name	Description
CPaintDC	For drawing in a window's client area (*OnPaint()* handlers only)
CClientDC	For drawing in a window's client area (anywhere but *OnPaint()*)
CWindowDC	For drawing anywhere in a window, including the nonclient area

- All devices have the same functions.
- Each class's ctor and dtor call the appropriate functions to get and release the Device Context.

Drawing With the GDI

Function	**Description**
MoveTo	Sets the current position in preparation for drawing
LineTo	Draws a line from the current position to a specified position and moves the current position to the end of the line
SetPixel	Draws a single pixel
Polyline	Connects a set of points with line segments
Ellipse	Draws a circle or an ellipse
Rectangle	Draws a rectangle with square corners

GDI Pens

Windows uses the pen that is currently selected into the device context to draw lines and borders.

MFC represents GDI pens with the class *CPen*.

```
CPen pen(PS_SOLID, 1, RGB(255, 0, 0));
CPen* pOldPen = dc.SelectObject(&pen);

// drawing using red pen

dc.SelectObject(pOldPen);
```

CPen uses three defining characteristics: style, width and color.

Note: the GDI object was deselected at the end of its usage.

The Mouse and the Keyboard

The two most common input devices are the mouse and the keyboard.

Their input comes in the form of messages.

The **`WM_`** messages, require further processing before being sent to the application.

Getting Input from the Mouse

Reported mouse events are:

- **WM_*x*BUTTON*action***

where *x*∈[L|M|R] and *action*∈[DOWN|UP|DBLCLK]

for left/middle/right button press/release/doubleclick.

// WM_MBUTTONDBLCLK on a middle button double click.

- **WM_MOUSEMOVE** for cursor movement.

A left double click creates the following messages:

WM_LBUTTONDOWN
WM_LBUTTONUP
WM_LBUTTONDBLCLK
WM_LBUTTONUP

Mouse Message Map

Message map macros and message handlers:

Message	***Message Map Macro***	***Function***
WM_LBUTTONDOWN	ON_WM_LBUTTONDOWN	*OnLButtonDown*
WM_MOUSEMOVE	ON_WM_MOUSEMOVE	*OnMouseMove*

Prototyped as follows:

```
afx_msg void OnMsgName(UINT nFlags,
                                 CPoint point);
```

Where ***point*** is the mouse location at the action and ***nFlags*** is the state of the mouse buttons, Shift and Ctrl keys.

Message Box

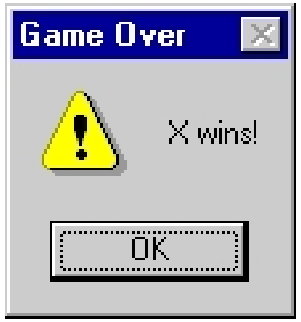
A useful tool to display messages:

```
int MessageBox(LPCTSTR lpszText,
  LPCTSTR lpszCaption=NULL, UINT nType=MB_OK);
```

For example:

```
MessageBox(_T("X wins!"), _T("Game Over"),
           MB_ICONEXCLAMATION | MB_OK);
```

Which returns with a **IDOK** code.



Game Over

X wins!

Getting Input from the Keyboard

A windows application learns of keyboard events just like the mouse, through messages.

For printable characters, a program receives a message whenever the key is pressed.

Just process the WM_CHAR message and check for key codes.

This message is mapped with an ON_WM_CHAR entry to *OnChar()*.

```
afx_msg void OnChar(UINT nChar,
               UINT nRepCnt, UINT nFlags);
```

Other virtual keys (Alt, ESC, PgUp, Left Arrow …) are handled differently.

Menus

Drop down menus are the most widely recognized GUI.

Menus contribute to programs ease of use.

Windows handles menus graphically, and MFC routes **menu item** commands to designated class members.

Menus resources can be placed in the resource file, and loaded on run time. The menu editing is done in Visual C++ resource editor.

A ***resource file*** (.rc) defines the applications resources, such as binary objects, menu, icons etc…

Menu Commands

WM_COMMAND messages are sent when selecting menu items (for example with an ID_FILE_NEW identifier).

An ON_COMMAND statement in the message map links this messages to a class member function.

```
ON_COMMAND(ID_FILE_NEW, OnFileNew)
```

Prototyped as

```
afx_msg void OnFileNew();
```

Updating the Items in a Menu

MFC provides a convenient mechanism for keeping menu items updated.

ON_UPDATE_COMMAND_UI macros in the message map link selected **member functions** to serve as *update handlers*.

```
ON_UPDATE_COMMAND_UI(ID_COLOR_RED,OnUpdateColorRed)
```

```
void CMainWindow::OnUpdateColorRed(CCmdUI* pCmdUI){
      pCmdUI->SetCheck (m_nCurrentColor == 0);
}
```

Other ***CCmdUI*** methods are *Enable()*, *SetCheck()*, *SetRadio()* and *SetText()*.

ON_UPDATE_COMMAND_UI connects **menu items** to **update handlers**.

Add Windows Message Handlers

Visual Studio's **ClassWizard** can be used to add *command handlers* and *update handlers*.

1. Right-click *CChildView* in the class view, and select 'Add Windows Message Handler'.
2. Select the ID_*identifier* from the 'Class or object to handle:'.
3. Double-click COMMAND or UPDATE_COMMAND_UI from the 'New Windows messages/events:'.
4. Accept the default function name.
5. Finish by clicking the Edit Existing button, to go to the handler.

Controls

A *control* is a special kind of window designed to convey information to the user or to acquire input.

The *classic controls* are:

Control Type	***WNDCLASS***	***MFC Class***
Buttons	"BUTTON"	*CButton*
List boxes	"LISTBOX"	*CListBox*
Edit controls	"EDIT"	*CEdit*
Combo boxes	"COMBOBOX"	*CComboBox*
Scroll bars	"SCROLLBAR"	*CScrollBar*
Static controls	"STATIC"	*CStatic*

The Classic Controls

MFC uses message maps to link control notifications to class member functions.

```
ON_BN_CLICKED(IDC_BUTTON, OnButtonClicked)
```

There are ON_EN macros for edit controls and ON_LBN macros for list box controls.

The generic ON_CONTROL macro handles all notifications and all control types.

Controls are windows! Some useful inherited ***CWnd*** member functions are *SetWindowText*, *SetFont*, *EnableWindow* etc…

Dialog Boxes

A *dialog box* is a window that pops up to obtain input from the user.

A *modal* dialog box disables its parent window until dismissed (like file open dialog)

A *modeless* dialog box acts as a conventional window. It does not disable its parent (like floating toolbar).

Both are encapsulated in MFC's *CDialog* class.

Use & to place an underline in the caption of a control.
"Save &As…" → Save As…

Creating a Modal Dialog Box

Create a new dialog using VC++ Insert Dialog command, from the Resource Tab.

Modify the dialog by adding desired controls.

Double Clicking the dialog opens the ClassWizard which guides to the addition of a new class, inherited from *CDialog*.

VC++ creates the new class, and places it in the project, linking it to the dialog box.

Activation of the new dialog class is done simple by:

```
CMyDialog dlg;
if ( dlg.DoModal() == IDOK ) {
      // The user clicked OK; do something !
}
```

Dialog Data Exchange (DDX)

A convenient way to expose the input from a dialog, is to map the control to a public member variable.

MFC’s *DoDataExchange()* uses DDX macros to transfer data between dialog’s controls and data members.

```
void CMyDialog::DoDataExchange(
                          CDataExchange* pDX){
     DDX_Text (pDX, IDC_NAME, m_strName);
     DDX_Text (pDX, IDC_PHONE, m_strPhone);
}
```

Linking two *CString* data members to edit controls.
The `pDX` argument is a pointer to a *CDataExchange* object.

Note: The linkage is **only** during entry and exit of the dialog.

DDX Functions

DDX Function	***Description***
DDX_Text	Associates a BYTE, an int, a short, a UINT, a long, a DWORD, a *CString*, a string, a float, a double, a *COleDateTime*, or a *COleCurrency* variable with an edit control.
DDX_Check	Associates an int variable with a check box control.
DDX_Radio	Associates an int variable with a group of radio buttons.
DDX_LBString	Associates a *CString* variable with a list box.
DDX_LBStringExact	Associates a *CString* variable with a list box.

Dialog Data Validation (DDV)

DDV allows MFC to validate the values entered into a dialog’s control before the dialog is dismissed.

The validation falls into two categories:

- Validate numeric variables so they fall within specified limits.
- Validate CString variable length to a certain limit.

```
void CMyDialog::DoDataExchange(
                      CDataExchange* pDX){
     DDX_Text (pDX, IDC_COUNT, m_nCount);
     DDV_MinMaxInt (pDX, m_nCount, 0, 100);
}
```

The DDV function call should immediately follow the DDX.

DDX and DDV from ClassWizard

ClassWizard can help in the addition of DDX and DDV's.

To add a DDX or DDV to a dialog box:

- Enter the Member Variable tab in the *ClassWizard* window.
- Select the dialog class's name in the Class Name box.
- Highlight the desired ID of the control, and click the Add Variable button.
- Type the member variable name and select the varibale type in the Add Member Variable dialog box.
- For numeric and string variables limiting, just add restrictions.

ClassWizard can be activated with Ctrl-W.

Interacting with other controls

It is possible to get a *CWnd* pointer to any control in a dialog box.

MFC’s *DDX_Control* function offers this capability.

```
DDX_Control(pDX, IDC_LIST, m_wndListBox);
```

Now adding strings to the list box is a simple:

```
m_wndListBox.AddString(_T(“One”));
```

Using ClassWizard to add DDX_Control is similar, with the difference of choosing **Control** instead of **Value** in the Add Member Variable dialog box.

Common Dialogs

Windows provides standard implementation of several common dialog boxes. MFC provides C++ interface to these classes.

Class	***Dialog Type(s)***
CFileDialog	Open and Save As dialog boxes.
CPrintDialog	Print and Print Setup dialog boxes.
CPageSetupDialog	Page Setup dialog boxes.
CFindReplaceDialog	Find and Replace dialog boxes.
CColorDialog	Color dialog boxes.
CFontDialog	Font dialog boxes.