

Large scale GUI app development in MS Windows / NT is often done with Microsoft Foundation Class library (MFC) using C++.

Windows Program

Windows is a message passing based system.

The entry to a windows program is **WinMain**. WinMain creates the application window and begins processing the applications messages via an event loop.

Messages are dispatched to be processed in the **windows procedure** (sent to message handler function).

Messages the application does not process are sent to **DefWindowProc** for default processing

Messages

Over 200 message types. All begin with **WM_***.

Message calls have 4 arguments:

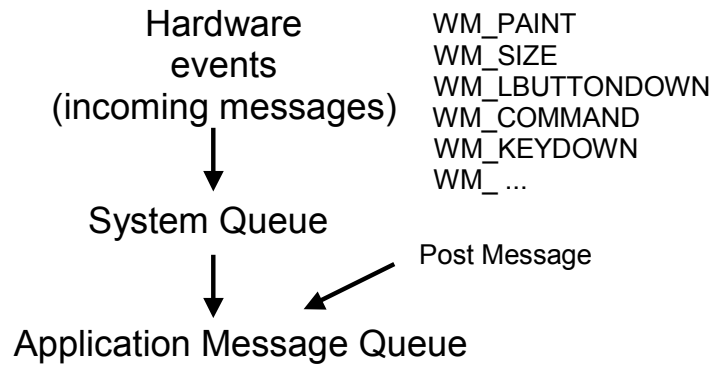
- handle to window message is directed to
- message ID
- wParam -- 32 bit word parameter
- lParam -- 32 bit long parameter

WM_LBUTTONDOWN message has:

- wParam bit flags describing state of ctrl, shift and mouse buttons.
- lParam has two 16 bit values for the location of the mouse pointer (when left button was down).

The message **WM_QUIT** stops the processing of the message loop and WinMain returns (exits). Clicking the close in the File menu or the upper left **X** button sends a **WM_QUIT** message.

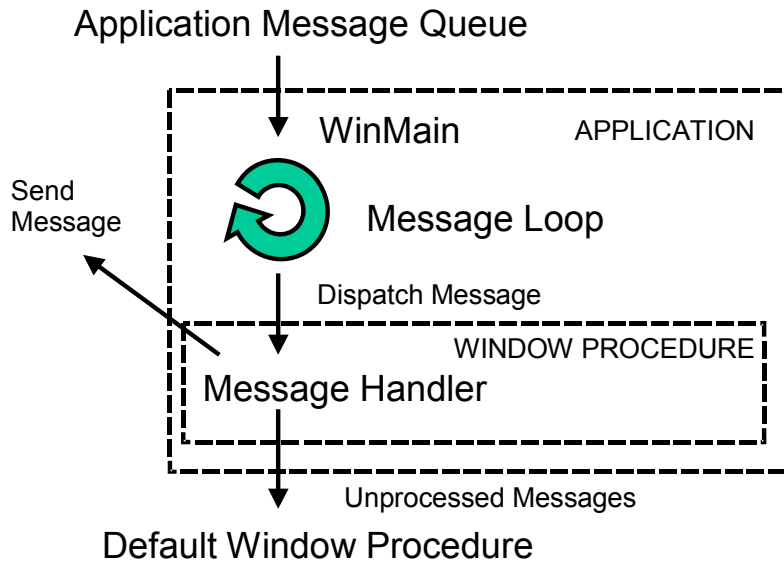
Windows programming model



Hungarian Notation: naming convention

All variables begin w/ type prefix

Prefix	Data Type	
b	bool	m_strWndClass
c	char	is a string
cx, cy	distance horizontal, vertical	member of WndClass
dw	DWORD	pszString
l	Long	is a pointer to zero-
m_	member of class	delimited string variable
n	int	
p	pointer	Strings
pt	CPoint or POINT	ANSI (8) Unicode (16)
rc	CRect or RECT	"hello" ANSI win
str	CString	L"hello" Unicode NT
sz	zero-delimited string	_T("hello")
w	WORD	preprocessor
wnd	CWnd	_UNICODE define
		generate Unicode



MFC application

Every win app created with MFC must have a global instance of **CWinApp**.

MFC links to WinMain.cpp that contains WinMain()

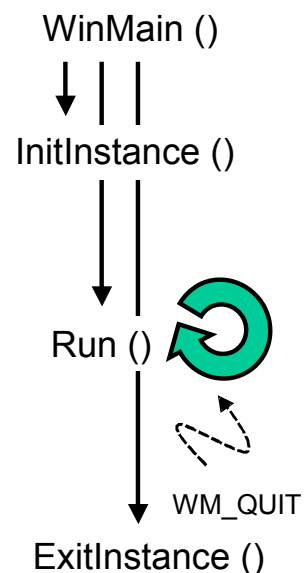
WinMain() calls the CWinApp instance's virtual function **InitInstance** that must be overridden in the application.

The application can initialize itself in InitInstance.

If InitInstance returns TRUE then WinMain calls CWinApp member function **Run** to process messages.

When **WM_QUIT** message terminates Run, WinMain calls CWinApp member function **ExitInstance** to clean up before termination.

an App



InitInstance ()
creates a new **CMainWindow**

set how to display the window
ShowWindow (arg)
SW_SHOWNORMAL,
SW_SHOWMINIMIZED

paints the window, **UpdateWindow**

Create a window

```
Create( window class, title, window style,  
        window size, parent window, menu name, extended style,  
        document/view create context);
```

Many Create (args) have default values for convenience.

Event processing w/ Message Maps

Message map is a complex macro solution to avoid each subclass of CCmdTarget having virtual functions for all the window messages it could receive. (This would require large vtables...)

Using Message Maps

Declare message map and all afx_msg functions in class declaration (*.h)

```
DECLARE_MESSAGE_MAP  
afx_msg void OnFunction();
```

In class implementation (*.cpp) create and initialize message map by placing macros identifying messages between calls to

```
BEGIN_MESSAGE_MAP  
....  
END_MESSAGE_MAP
```

Add member functions to handle the messages (afx_msg functions)

Windows WM_MESSAGE names are renamed in message maps as ^{MFC}ON_WM_MESSAGE 9

The name of handlers becomes *OnMessage*. Where all but the first letters of words are lowercase.

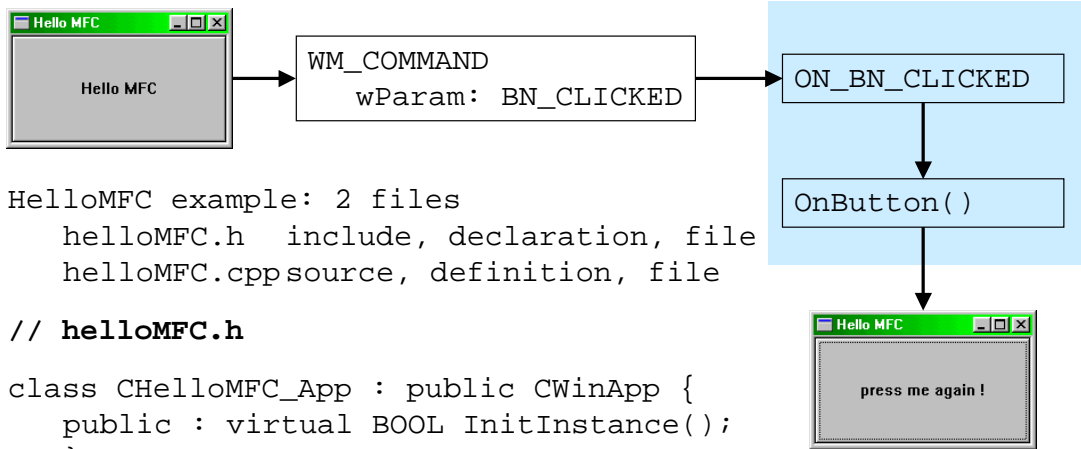
Message	Map Macro	function called
WM_CREATE	ON_WM_CREATE()	OnCreate()
WM_PAINT	ON_WM_PAINT()	OnPaint()
WM_SIZE	ON_WM_SIZE()	OnSize(...)

Control classes have macros to simplify WM_COMMAND

```
ON_BN_CLICKED(IDC_BUTTON, OnButton)
    OnButton() {...}
```

<< do HelloMFC walkthrough here >>

```
ON_CONTROL_RANGE(BN_CLICKED, IDC_RED, IDC_GREEN,
    OnButton)
    OnButton(UINT nID) { ... }
```



HelloMFC example: 2 files
helloMFC.h include, declaration, file
helloMFC.cppsource, definition, file

```
// helloMFC.h
```

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

```
// declare members, message handlers, message map
```

```
class CMainWindow : public CFrameWnd {
public : CMainWindow(); // construct frame
protected : afx_msg int OnCreate(LPCREATESTRUCT);
             afx_msg void OnButton();
             DECLARE_MESSAGE_MAP()

private : CButton m_ctlButtonW;
};
```

```
# include <afxwin.h>
# include "HelloMFC.h"

// set value for button resource
# define IDC_BUTTON      100

CHelloMFC_App helloApp; // create an application
framework

// Must redefine InitInstance to initialize application
BOOL CHelloMFC_App :: InitInstance () {
    m_pMainWnd = new CMainWindow;
    m_pMainWnd->ShowWindow (m_nCmdShow);
    m_pMainWnd->UpdateWindow();
    return TRUE;      }

// message map for event routing
BEGIN_MESSAGE_MAP (CMainWindow, CFrameWnd)
    ON_WM_CREATE ()
    ON_BN_CLICKED (IDC_BUTTON, OnButton)
END_MESSAGE_MAP ()
```

```
// constructor for MainWindow
CMainWindow :: CMainWindow() {
    CRect rect = new CRect(0,0,200, 123);

    // Window Style WS_OVERLAPPEDWINDOW overlaps others,
    // minimize and maximize button, border, and system menu
    Create (NULL, "Hello MFC", WS_OVERLAPPEDWINDOW, rect);
}

// message handlers

// MainWindow exists now create child controls
int CMainWindow :: OnCreate(LPCREATESTRUCT lpcs) {
    CRect rect;

    GetClientRect(&rect); // get size of MainWindow
    // create pushbutton as a control member
    m_ctlButtonW.Create("Hello MFC",
        WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON,
        rect, this, IDC_BUTTON);
    return 0;
}
```

```
// handle button press events
void CMainWindow :: OnButton() {
    static int toggle = 0;

    toggle++;
    // uncomment the next line to see MessageBox afx use
    // MessageBox ("Button was pressed");
    if (toggle % 2)
        m_ctlButtonW.SetWindowText("press me again !");
    else
        m_ctlButtonW.SetWindowText("Hello CS585");
}
```

Benefits of C++ and MFC

OO representation of Windows API and application: windows functions look like become class member functions, windows resources are data members.

Assertion class library: `ASSERT(expr)` and `ASSERT_VALID(object)`

MFC Classes

CObject root object provides: persistence objects (serialization), runtime class support (is-a type info), debugging and robust code support.

CCmdTarget encapsulates all message processing actions. `CWinApp` derived from `CCmdTarget`.

CWinApp represents applications.

- ProcessShellCommand - command line args

- OnFileOpen -- file menu's open command

- InitInstance (vfn - override for startup up actions)

CWnd encapsulates a window -- many default message handlers & member functions for windows API. Base class for other windows:

CFrameWnd represent top level windows.

CDialog encapsulates dialog windows

User interface controls {**CButton**, **CListBox**, **CEdit**, **CComboBox**, **CScrollBar**, **CStatic**}

GDI Classes

Graphics Device Interface does screen, printer, other devices.

CDC encapsulates window's device context -- e.g. for drawing operations {**DrawText**, **LineTo** .. using a **CPaintDC**, **CClientDC**, **CWindowDC**.

Drawing to a device requires the use pens, brushes..

CGdiObjects is the base class for **CPen**, **CBrush**, **CBitmap**, **CPalette** ...

Miscellaneous classes

Classes (and templates) for data structures: **CPtrArray**, **CStringArray**, **CStringList**, **CObList**, ...

CPoint (point), **CRect** (rectangle), **CTime**, ...

File, database access, OLE, thread classes.

Afx functions (application framework extension)

MFC also provides global functions (available outside context of class)

e.g. **AfxAbort**

MessageBox is easy

All window classes inherit **MessageBox** from **CWnd**

```
int MessageBox(LPCTSTR text, LPCTSTR title = null,
              UINT style = MB_OK);
```

```
MessageBox("Click OK to continue", "This App");
```

```
MessageBox("Save data?", "Title", MB_YESNOCANCEL);
```