SFU CMPT-212 2008-1 Topic: Windows GUI Applications

Ján Maňuch

E-mail: jmanuch@sfu.ca

Thursday 27th March, 2008

GUI vs. Console applications

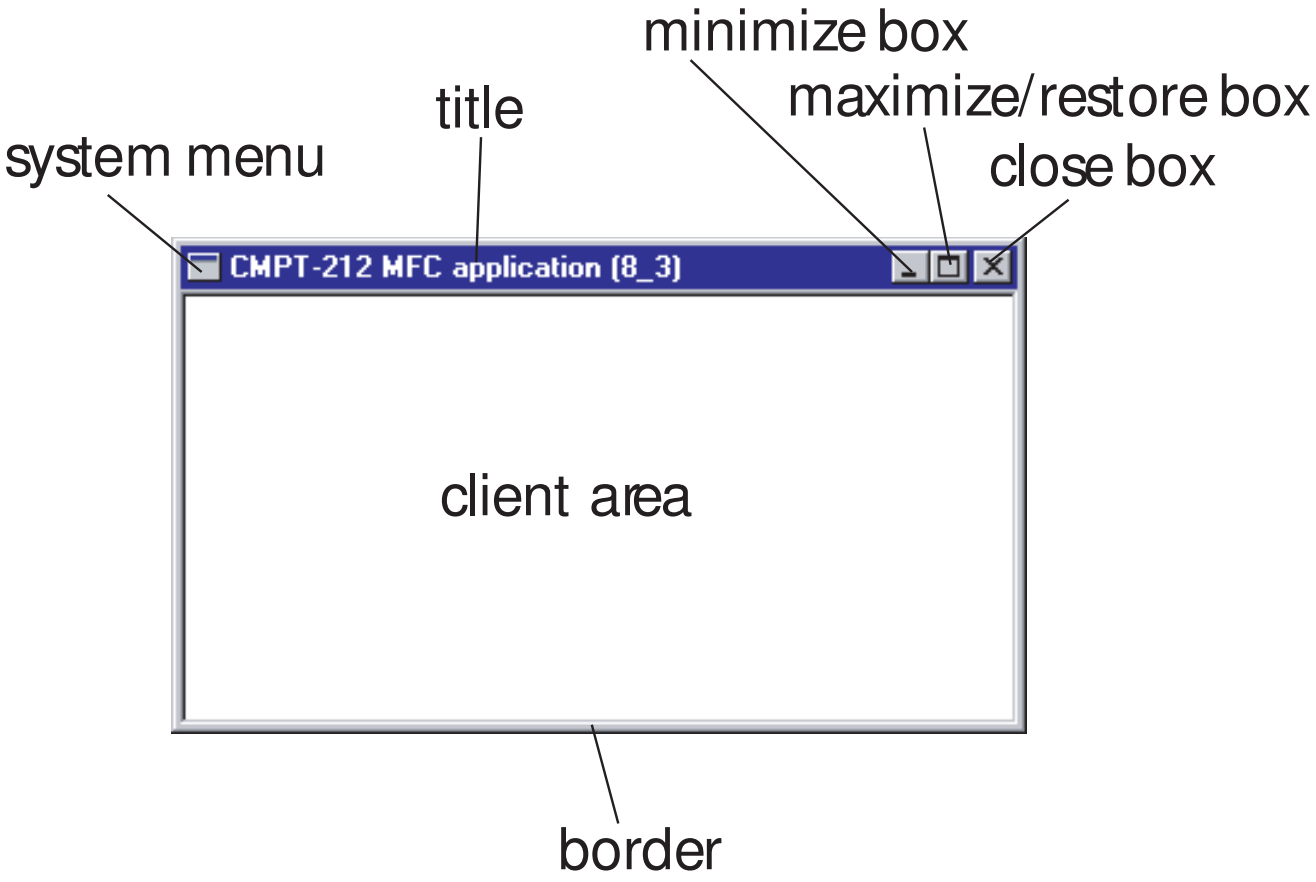
- *Graphical User Interface* (GUI) applications represent a different programming paradigm than **console applications**.
- Console applications: **structured commands, data received through console input**
  GUI applications: react to *asynchronous user and environment events*

The Windows API

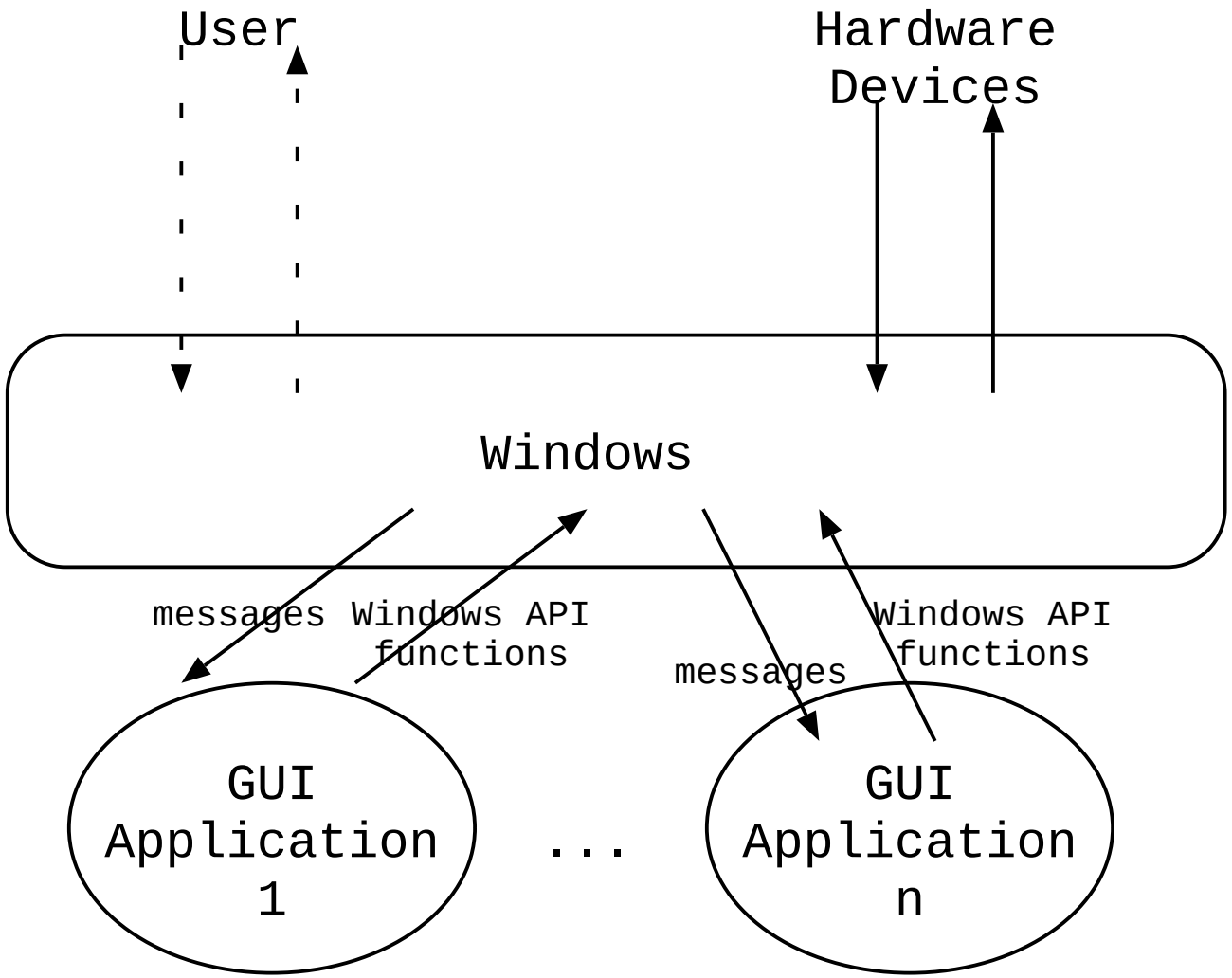
- The Windows Application Programming Interface™ (API) is a collection of libraries, macros, types, global variables, functions and documentation, that is necessary for writing GUI applications for the Microsoft© Windows™ family of operating systems.

- The Windows API provides the **lowest level** of Windows GUI functionality to programmers.
- The Windows API has a common core, and specific variations for different operating systems: Windows™ 3.1, NT™ , Windows 95™ , Windows 98™ , Windows NT™ , Windows 2000™ , Windows XP™ , Windows CE™ , etc.
- The Windows API hides **major differences between operating systems**.

Components of a Window



Windows GUI Applications



Convention in naming variables

Windows API and MFC programmers use Hungarian notation prefixes in naming variables:

- `ar` – array
- `b` – `BOOL` or `bool`
- `c` – `char`
- `C` – class
- `d` – `double`
- `l` – `long`
- `lp` – `long` pointer
- `m_` – class member variable
- `n` or `i` – integer
- `p` – pointer
- `s` – string
- `sz` – zero terminated string
- `s_` – static class member variable

Elements of a Windows GUI application

WinMain: `WinMain()` is the *entry point* of every Windows application (as `main()` is the entry point of every console application).

```
1  int WINAPI WinMain(
2    HINSTANCE hInstance,     // handle to current instance
3    HINSTANCE hPrevInstance, // handle to previous instance
4    LPSTR lpCmdLine,         // command line
5    int nCmdShow )           // show state
6  {
7    // create "window class" of type WNDCLASS
8    // set parameters of window class, namely
9    //   the callback function
10   // register window class
11   // create and show main window
12   // execute message loop
13 }
```

Window class: defines the "class" of windows displayed by this appl.:

- we have to specify things like: the program instance handle HINSTANCE, the mouse cursor, the brush to paint the window's background, and the symbolic name of our class.

```
1 WNDCLASS wincl; // Data structure for the windowclass
2 wincl.hInstance = hInstance;
3 wincl.lpszClassName = "CMPT-Application";
4 // ..etc..
```

- The most important part of the Window Class is the address of the Callback procedure, or the Window Procedure. Windows is supposed to call us — Windows sends messages to our program by calling this procedure.

```
1 wincl.lpfnWndProc = WindowProcedure;
```

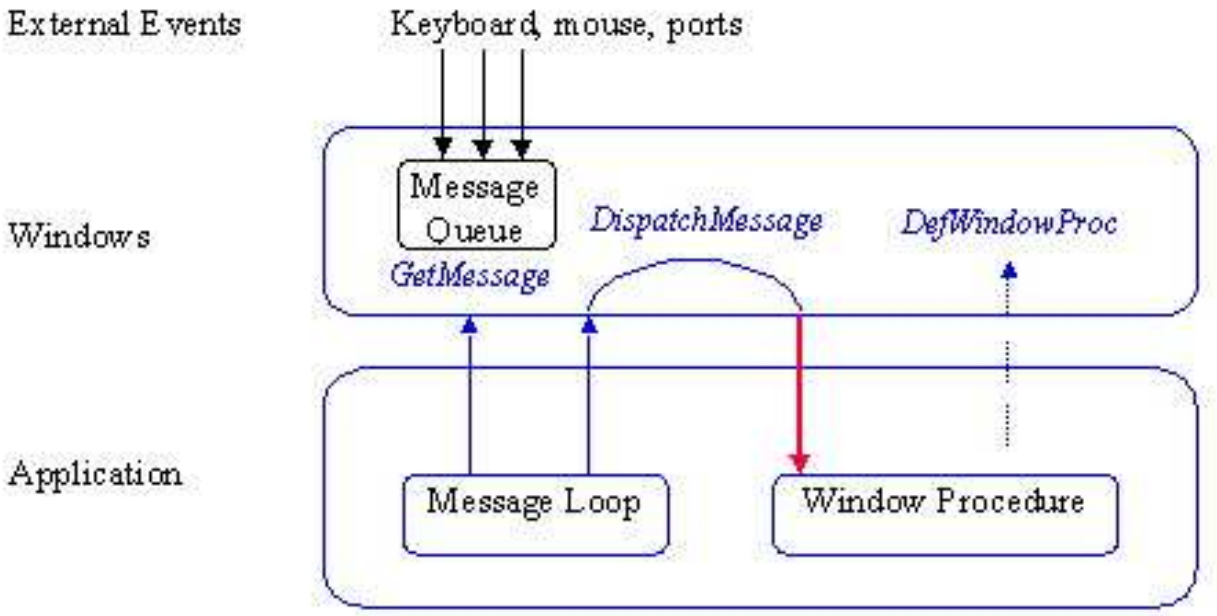
- Once all the fields of WNDCLASS are filled, we register the class with the Windows system.

```
1 if (!RegisterClass(&wincl))
2   return 0;
```

Main Window: Most useful GUI applications have a main application window as the *primary GUI element* — created by `CreateWindow()` procedure:

```
1  HWND hwnd = CreateWindow ( // returns handle to the window
2    "CMPT-Application",  // Classname - has to be registered!
3    "Windows App",       // Title Text
4    WS_OVERLAPPEDWINDOW, // default window
5    CW_USEDEFAULT,       // Windows decides the position
6    CW_USEDEFAULT,       // of the window on the screen
7    544,                 // The programs width
8    375,                 // and height in pixels
9    HWND_DESKTOP,        // a child-window to desktop
10   NULL,                // No menu
11   hInstance,           // Program Instance handler
12   NULL                 // No Window Creation data
13 );
```

Windows messages: Hardware events or requests from the user (hardware events interpreted as . . .) destined for the application result in *messages sent to the main window of the application*.



Message Loop: Every application must contain a message loop, where all messages destined to the application are *dispatched*.

```
1  while (GetMessage (&messages, NULL, 0, 0)) {
2    /* Translate virtual-key messages into
3       character messages */
4    TranslateMessage(&messages);
5    /* Send message to WindowProcedure */
6    DispatchMessage(&messages);
7  }
```

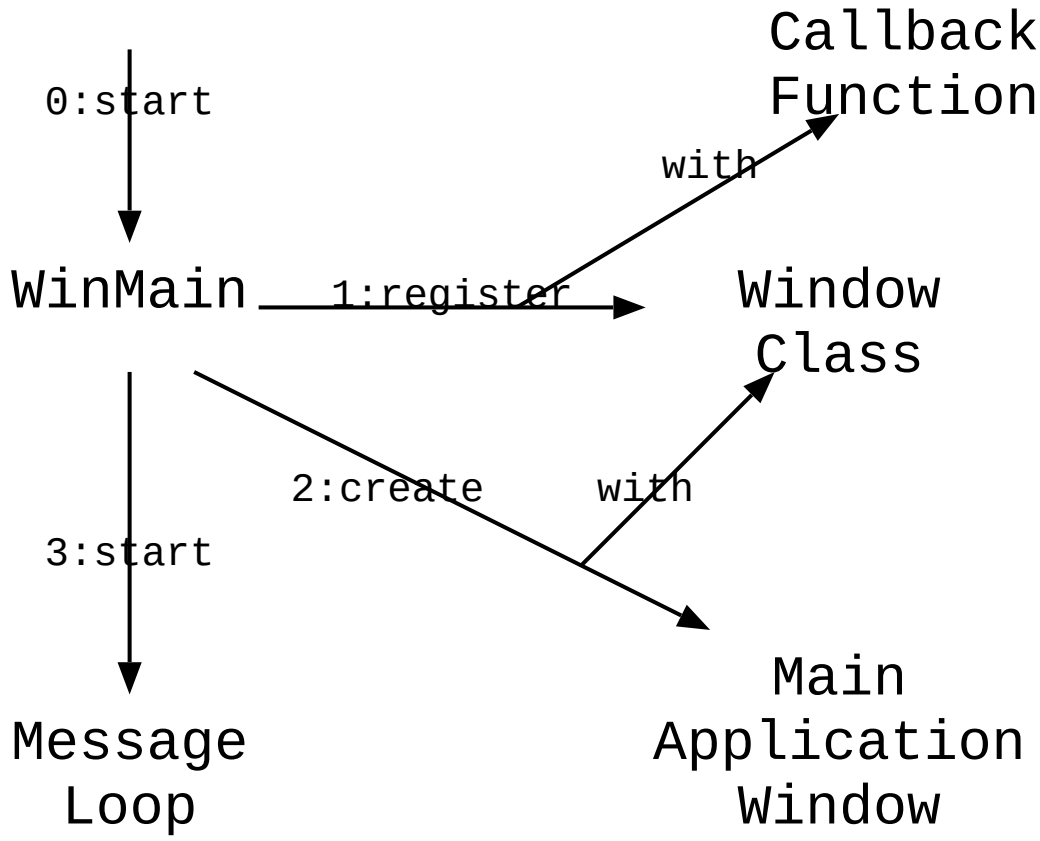
Callback function: Every window *reacts to requests or changes in the environment* through code in a callback function (a function called by Windows), often called `WindowProcedure()`:

```
1  LRESULT CALLBACK WindowProcedure(
2    HWND hWnd,      // handle to window
3    UINT uMsg,      // message identifier
4    WPARAM wParam,  // first message parameter
5    LPARAM lParam ) // second message parameter
```

```
6   {
7     switch( uMsg ) {
8     case WM_DESTROY:
9       PostQuitMessage( 0 );
10      // send a WM_QUIT to the message queue
11      break;
12    default:
13      return DefWindowProc( hWnd, uMsg, wParam, lParam );
14      // for messages that we don't deal with
15    }
16    return 0;
17  }
```

Note (Windows API functions)**.** *Every application can request services from the operating system (including sending messages to itself !!)* ***only through calls*** *to Windows API functions.*

Start of a Windows GUI application

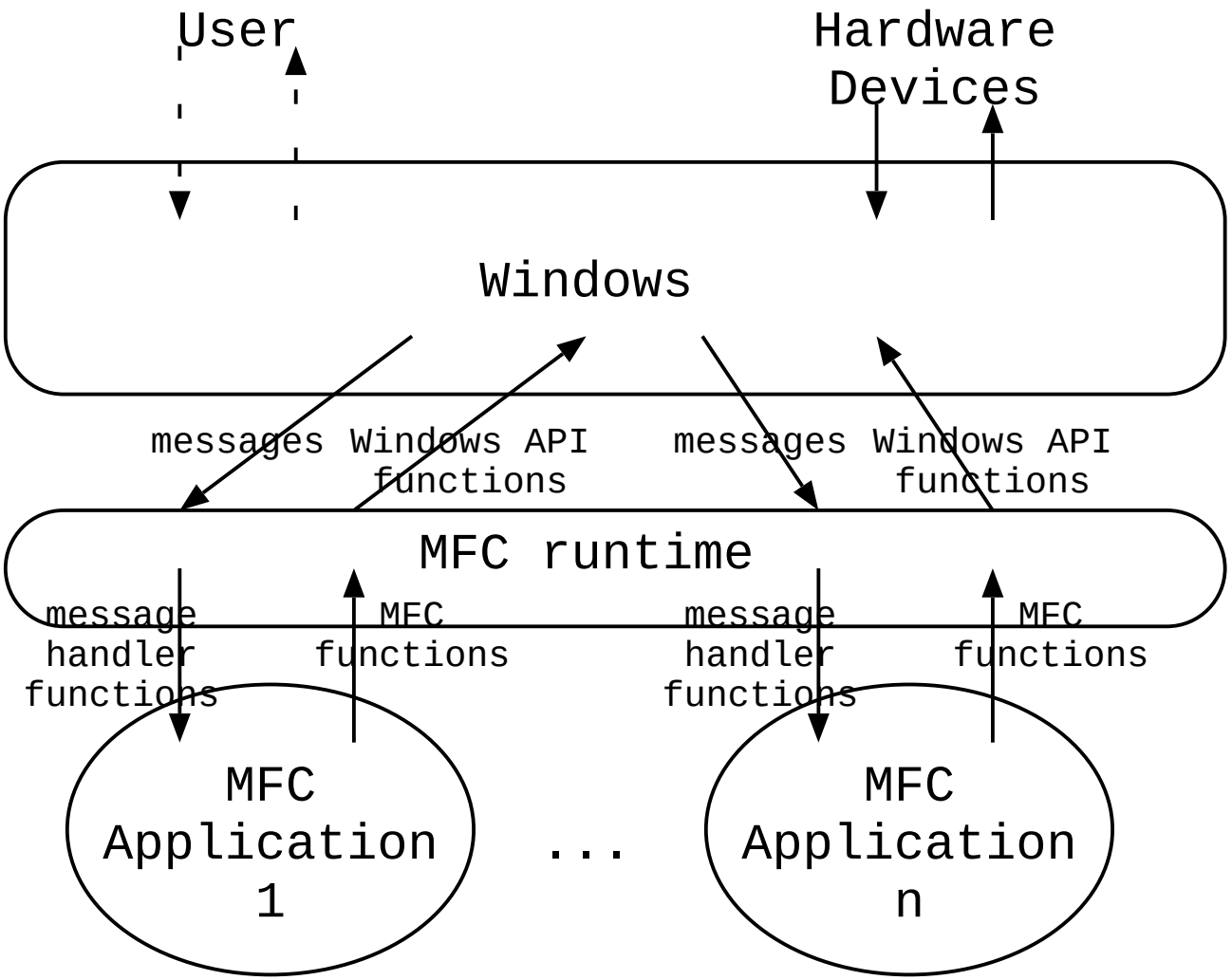


Remark: Main application window and Window class share the class name and the program instance handle.

Example: winapi.cpp, without comments: winapicode.cpp

MFC Applications

- The Windows API is **not object-oriented**, but is *object-based*, i.e. most entities can be thought of as objects, and most functions refer to objects. A basic *typing polymorphism* is also supported.
- Microsoft Foundation Classes (MFC) are a C++ *object-oriented wrapper* to the Windows API.
- MFC provides:
  - a more intuitive way of *thinking* about Windows GUI applications.
  - reduction in the size of custom code, corresponding to many repetitive programming tasks.


User
Hardware Devices
Windows
messages
Windows API functions
messages
Windows API functions
MFC runtime
message handler functions
MFC functions
message handler functions
MFC functions
MFC Application 1
...
MFC Application n

Elements of an MFC application

Note (WinMain)**.** *`WinMain()` is still the **only** entry point of MFC applications.* `WinMain()` is provided by the MFC runtime.

Note (Callback functions)**.** *Windows declared as MFC objects have* implicit callback functions *associated with them. There is no need to declare any.*

Note (Message Loop)**.** *The MFC runtime* provides the message loop. *`CWinApp::Run()` can be overloaded to provide custom processing in the message loop.*

Note (Windows messages)**.** *The MFC runtime* processes Windows messages.

Elements of an MFC application

Note (CWinApp)**.** *Stand-alone MFC applications must declare an instance of* `CWinApp`*, or of a derived class.*

```
1 class CApplication : public CWinApp
2 { ... };
3 CApplication app_212;
```

Note (Main Window)**.** *Stand-alone MFC applications must create a main window object and assign its pointer to* `CWinApp::m_pMainWnd`*. The main window must be derived from* `CWnd` *(for example, it can be* `CFrameWnd`*).*

```
1 m_pMainWnd = new CFrameWnd;
```

Example: firstmfc.cpp, the code without comments: firstmfccode.cpp

Sample questions

- What is the “main” function of Windows application? What is this function supposed to do?
- Explain what’s happening with messages generated in Windows for different applications.
- Why is it important to provide *Callback function* when registering windows class?
- Which elements (code) of API applications is provided automatically in MFC application? What must an MFC application contain?

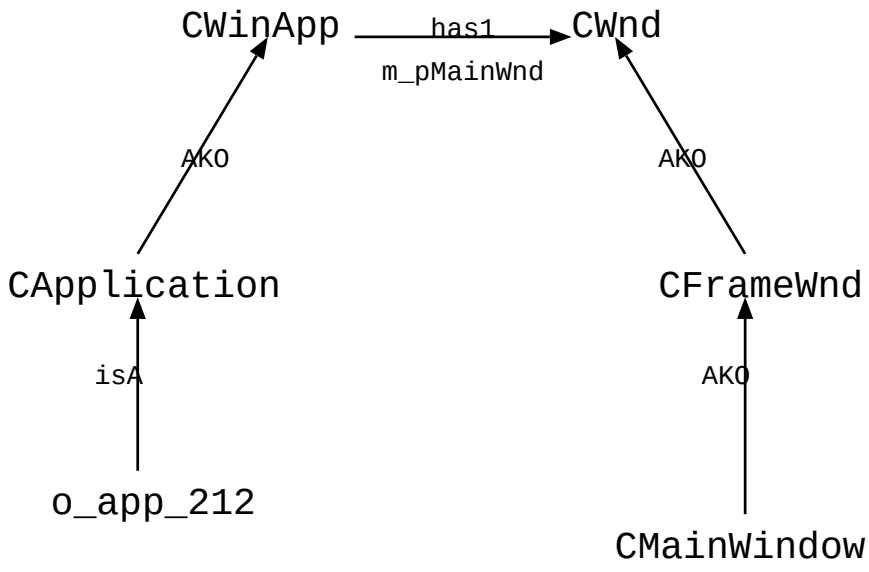
MFC - customizing main window

it's usual to derive own class from `CWnd` *or any of its subclasses*

- tasks connected with construction of the window are then performed in the constructor of this class
- when the window is destroyed, the desctuctor should clean-up
- it also allows to add message handlers

Example: firstmfcagain.cpp

Objects in a MFC application:



Note (Message handlers)**.** *Message handlers are MFC window class member functions with automatic names.*

Note (Message map)**.** *The association between a Windows message and a certain message handler is done through* message map macros.

```
1 BEGIN_MESSAGE_MAP( CMainWindow, CFrameWnd )
2   ON_WM_DESTROY()
3   ON_COMMAND(IDM_THINK, onThinking)
4 END_MESSAGE_MAP()
```

- for system defined messages: line 2 will associate the message `WM_DESTROY` with the message handler `onDestroy()` of class `CMainWindow`
- for user defined messages: line 3 will associate the message `IDM_THINK` with the message handler `onThinking()` of class `CMainWindow`

Example: ondestroy.cpp

Note (MFC member functions)**.** *MFC applications generally request services from the operating system through calls to member functions of objects.*

Example:

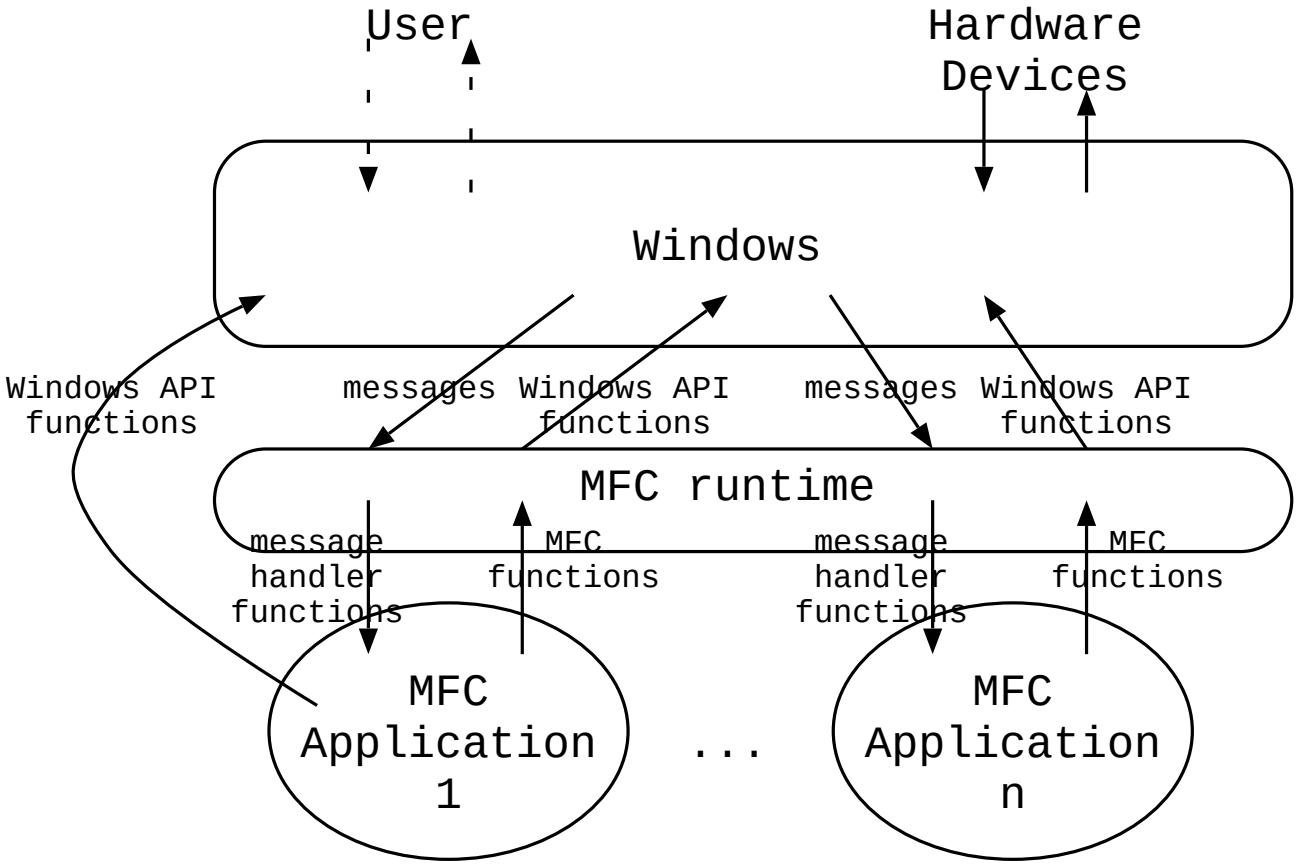
```
1 SendMessage(WM_DESTROY);
2 MessageBox("Quitting","Message:");
```

Note (Windows API functions)**.** *MFC applications can **also request services** from the operating system through calls to Windows API functions:*

- *MFC applications can call Windows API functions directly, without using wrappers.*
- *useful for the Windows API functions that have no MFC wrappers (e.g. Multimedia).*

Example: play.cpp

MFC applications and the Windows API



Sample questions

- How does an MFC application deal with messages from Windows?
- What's the difference between Windows API functions and MFC functions?