Windows Resources

- An executable file, may use a number of resources such as:
  - Icons,
  - Bitmaps,
  - Strings,
  - Any other binary data.
- The resources are embedded (linked) in the executable file.
- The resources are defined using a resource script file.
- The script file is compiled using the Windows resource compiler tool rc.exe.

The Resource Script File

- A resource script file is a text file with the RC extension.
- For example:

```
MyIcon      ICON   my.ico
MyCursor    CURSOR my.cur
MyBitmap    BITMAP my.bmp
```

- When the file is compiled, the result is emmitted to a file with the RES extension that may be linked to your application.

Example: Loading an Icon

- When creating a window class we had:

```
wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
```

- IDI_APPLICATION is a constant telling Windows to use the default Windows icon for windows derived from this class.
- If we want to use an icon specified in a resource file, we use:

```
wc.hIcon = LoadIcon(NULL, "MyIcon");
```


Loading Bitmaps and Cursors

```
HBITMAP LoadBitmap(HINSTANCE hInstance,
                   LPCTSTR lpBitmapName);

HCURSOR LoadCursor(HINSTANCE hInstance,
                   LPCTSTR lpCursorName);
```

Creating Menus (Resource File)

```
#include "Resource.h"

MyMenu MENU
{
  POPUP "&File"
  {
    MENUITEM "&New",      IDM_FILENEW
    MENUITEM "&Open...", IDM_FILEOPEN
    MENUITEM SEPARATOR
    MENUITEM "E&xit",     IDM_FILEEXIT
  }
  POPUP "&Help"
  {
    MENUITEM "&About",    IDM_ABOUT
  }
}
```


Creating Menus (Resource.h)

- Here we define the constants used by our resource file:

```
#define IDM_FILENEW  1000
#define IDM_FILEOPEN 1001
#define IDM_FILEEXIT 1002
#define IDM_ABOUT    1003
```

- By conventions menu item constants start with IDM_ (For icons it is IDI_, etc…)

Using the Menu

- Now, when creating the window class, we specify the resource name for the menu:

```
wc.lpszMenuName = "MyMenu";
```

- The menu events are handled in the WM_COMMAND message send to the callback.
- The wParam value of the message will tell us, the ID of the menu item clicked (i.e. match against the IDM_XXX constants).
- Actually the LoWord of wParam tells us the ID, by the HiWord is zero if the message comes from a menu.

Using Accelerators (1)

- An accelerator table must be defined in the resource file. Like:

```
MyAccelTable ACCELERATORS
BEGIN
   "^C",  IDDCLEAR         ; control C
   "K",   IDDCLEAR         ; shift k
   "k",   IDDELLIPSE, ALT  ; alt k
   98,    IDDRECT, ASCII   ; b
   66,    IDDSTAR, ASCII   ; B (shift b)
   "g",   IDDRECT          ; g
   "G",   IDDSTAR          ; G (shift G)
   VK_F1, IDDCLEAR, VIRTKEY                ; F1
   VK_F1, IDDSTAR, CONTROL, VIRTKEY        ; control F1
   VK_F1, IDDELLIPSE, SHIFT, VIRTKEY       ; shift F1
   VK_F1, IDDRECT, ALT, VIRTKEY            ; alt F1
   VK_F2, IDDCLEAR, ALT, SHIFT, VIRTKEY    ; alt shift F2
   VK_F2, IDDSTAR, CONTROL, SHIFT, VIRTKEY ; ctrl shift F2
END
```

Using Accelerators (2)

- To load the accelerator table from the resource file, use:

```
HACCEL LoadAccelerators(HINSTANCE hInstance,
                        LPCTSTR   lpTableName);
```

- The normal window message loop was:

```
while(GetMessage(&Msg, NULL, 0, 0) > 0)
{
  TranslateMessage(&Msg);
  DispatchMessage(&Msg);
}
```


Using Accelerators (3)

- To handle accelerators, it must be modified to:

```
while(GetMessage(&Msg, NULL, 0, 0) > 0)
{
  if (TranslateAccelerator(hWnd, hAccel, Msg) == 0)
   {
    TranslateMessage(&Msg);
    DispatchMessage(&Msg);
   }
}
```

- In other words, the normal Translate/DispatchMessage calls must not be mage if a accelerator was translated (see TranslateAccelerator on MSDN).

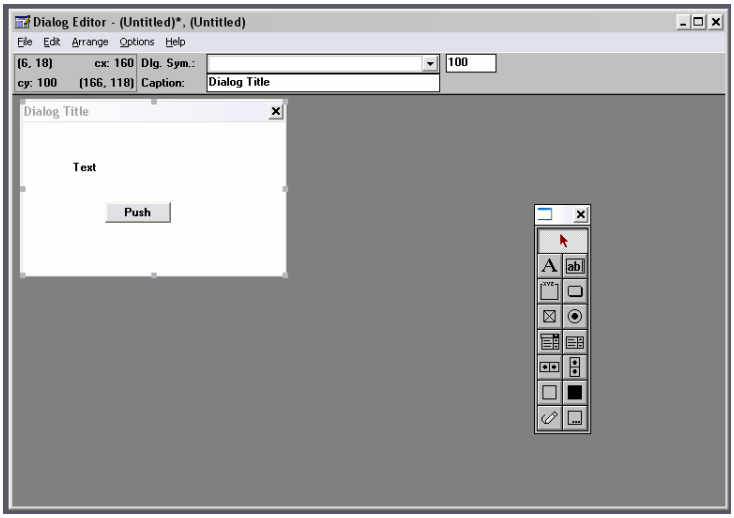
Using Accelerators (4)

- Again, the fact that an accelerator was pressed, is detected in the WM_COMMAND part of the callback.
- As usual, the LoWord of wParam will tell us the ID of the accelerator pressed, and the HiWord will be 1 so we can distinguish it from a menu item click.
- Note:
  - This means that the constant used to identify a menu and the constant to identify an accelerator may be the same. We can have a menu item “Save” and the combination “CTRL+S” handled by the same piece of code!

Dialogs

- Another resource can be a windows dialog.
- It is likely that you will use the dialog editor to create the dialogs rather than write the script manually.
- To create the dialog in your application you will call the CreateDialog API call.
- This function
  - Returns the hWnd of the dialog window.
  - You also specify (apart from others) the pointer to the callback function of the dialog.

The Dialog Editor



The Dialog Callback

- The prototype of the dialog callback is the same as the prototype of a “normal” windows callback.
- The messages are different.
- For example, when the dialog is created, we use the WM_INITDIALOG message rather than WM_CREATE.
- You should check out the MSDN entry for CreateDialog for more details.

What is COM?

- COM – **Component Object Model**.
- Platform Independent.
- Object Oriented.
- Binary Component Format.
- COM is a STANDARD specifying how objects are accessed and how they interact with other objects.
- The only requirements for a language to support com is the ability to create and manipulate pointers and calling functions through pointers.

More COM

- A software component is made up of:
  - **Object Data** (Variables/Properties).
  - Data manipulated by a number of **functions**.
- The set (list) of functions is called the **interface**.
- These functions are implemented as **methods**.
- COM specifies that the only way to call a function is through a pointer to the interface.
- In COM there are a number of interfaces that must be implemented by all components.

Interfaces

- An interface is usually thought of as a **contract** specifying a group of related function prototypes:
  - Their name, return types and arguments.
- No implementation is associated with an interface.
- For example if we have an IQueue interface (by convention interface names start with the letter I) that defines the functions:
  - Enqueue
  - Dequeue

Accessing the Component

- An instance of an interface is a pointer to an array of function pointers.
- Each interface is assigned a Globally Unique Identifier (GUID).
- Note that one object may have multiple interfaces.
- COM interfaces are **immutable** – you cannot change the interface once it is assigned a GUID.

Registering COM Components

- The Windows Registry is a global system database for the OS.
- When a component is registered the GUID and the actual COM EXE or DLL are saved in the registry.
- Whenever we need to access a component (which we identify by the GUID), the registry is consulted to resolve the GUID into the COM EXE or DLL required.
- The component will be loaded in-process or out-of-process as required.

COM Clients and Servers

- A **COM Client** is the actual software that gets the interface pointer to an object and calls the methods.
- A **COM Server** where the interface implementations actually exist. The client gets a pointer to the interface to call the functions.
- There are 2 types of servers:
  - In-process: Implemented as DLLs.
  - Out-of-process: Implemented as EXEs. The process can be run on a different machine.