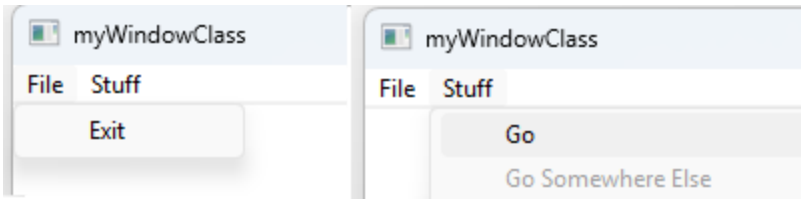
Menus and Icons

Adding a Menu requires a special piece of code within the `.rc` file

For example the code

```
#define IDR_MYMENU                  101
#define IDI_ICON1                   102
#define IDI_ICON2                   103
#define ID_FILE_EXIT                40001
#define ID_STUFF_GO                 40002
#define ID_STUFF_GOSOMEWHEREELSE    40003
```

```
#include "resource.h"
.
.
.
IDR_MYMENU MENU DISCARDABLE
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "E&xit",                   ID_FILE_EXIT
    END
    POPUP "&Stuff"
    BEGIN
        MENUITEM "&Go",                     ID_STUFF_GO
        MENUITEM "Go &Somewhere Else",      ID_STUFF_GOSOMEWHEREELS
E, GRAYED
    END
END
```

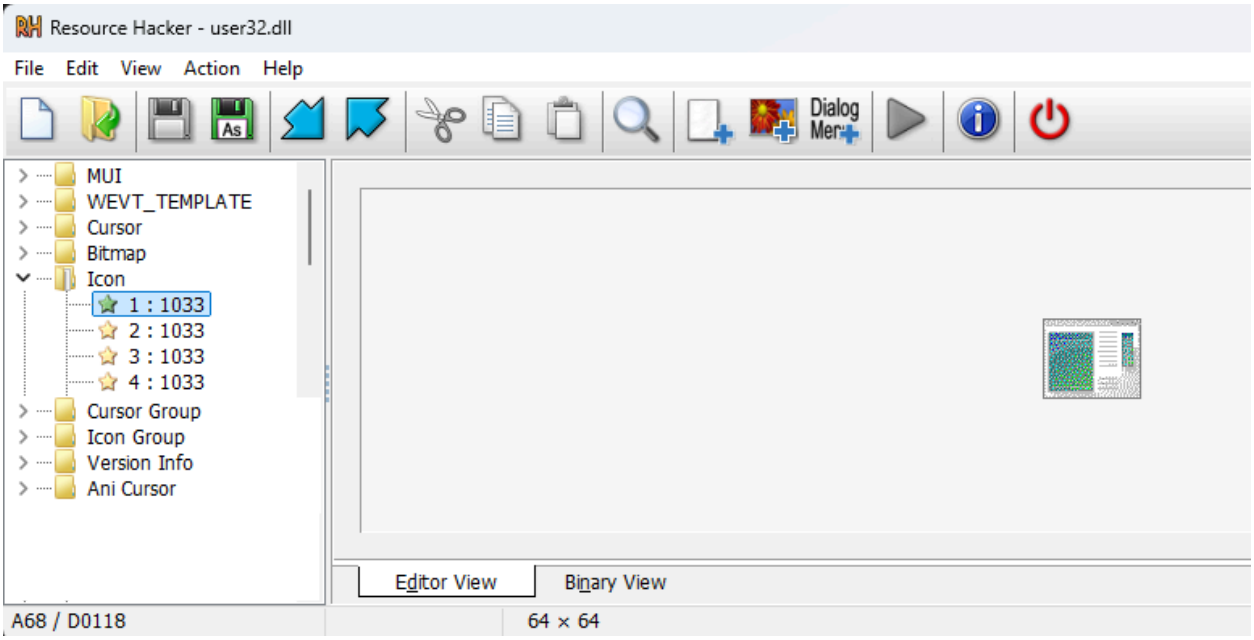


Important keywords here are defined in the header file.

Icons, cursors and other stuff can be initialized via the windows class parameters before registration. Like

```
wcex.hCursor = LoadCursor(nullptr, IDC_ARROW);
wcex.hIconSm = LoadIcon(nullptr, IDI_APPLICATION);
```

These points to pre-defined resources that can be found within "C:\Windows\System32\user32.dll"



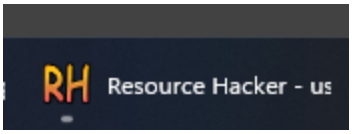
As well Icons can be set as a response to the message `WM_CREATE`

```
        hIcon = reinterpret_cast<HICON>(LoadImage(NULL, "Icon1.ico", IMAGE_I
CON, 32, 32, LR_LOADFROMFILE));
        if (hIcon)
            SendMessage(hwnd, WM_SETICON, ICON_BIG, (LPARAM)hIcon);
        else
            MessageBox(hwnd, "Could not load large icon!", "Error", MB_OK | MB_I
CONERROR);


        hIconSm = reinterpret_cast<HICON>(LoadImage(NULL, "Icon2.ico", IMA
GE_ICON, 16, 16, LR_LOADFROMFILE));
        if (hIconSm)
            SendMessage(hwnd, WM_SETICON, ICON_SMALL, (LPARAM)hIconS
m);
        else
            MessageBox(hwnd, "Could not load small icon!", "Error", MB_OK | MB_I
CONERROR);
```

Here you can set the small window icon

And the bigger one



See **SendMessage function (winuser.h)**

Menu Set up

The menus can also be set as a response to `VM_CREATE` message

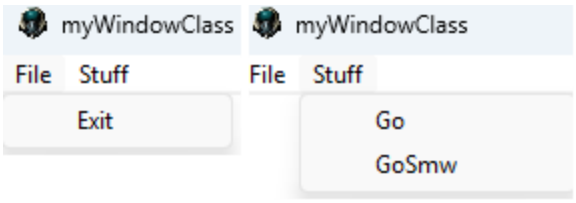
```
        HMENU hMenu, hSubMenu;
        HICON hIcon, hIconSm;

        hMenu = CreateMenu();

        hSubMenu = CreatePopupMenu();
        AppendMenu(hSubMenu, MF_STRING, ID_FILE_EXIT, "E&xit");
        AppendMenu(hMenu, MF_STRING | MF_POPUP, (UINT)hSubMenu, "&Fil
e");

        hSubMenu = CreatePopupMenu();
        AppendMenu(hSubMenu, MF_STRING, ID_STUFF_GO, "&Go");
        AppendMenu(hSubMenu, MF_STRING, ID_STUFF_GOSOMEWHEREELSE,
"&GoSmw");
        AppendMenu(hMenu, MF_STRING | MF_POPUP, (UINT)hSubMenu, "&Stu
ff");

        SetMenu(hwnd, hMenu);
```



Menu Actions

Menu Actions can be arranged as a response to `VM_COMMAND` message

```
    case WM_COMMAND:
        switch (LOWORD(wParam))
        {
```

```
        case ID_FILE_EXIT:
            PostMessage(hwnd, WM_CLOSE, 0, 0);
            break;
        case ID_STUFF_GO:
            MessageBox(hwnd, "You clicked Go!", "Woo!", MB_OK);
            break;
        }
        break;
```

Here I have two actions, one for EXIT where `VM_CLOSE` message is called and a another one for the `Go` button where a Message box is created.