Win API vs. GLUT

- OpenGL applications can be written using both Windows API or GLUT
- GLUT
  - Makes it possible to produce one source code for all operating systems supporting OpenGL
  - Has simple but enough window interface (mouse and keyboard input, windows, menus, etc.)
- Windows API
  - Offers so called WGL to port 3D OpenGL API into 2D Windows API
  - Gives all Windows features

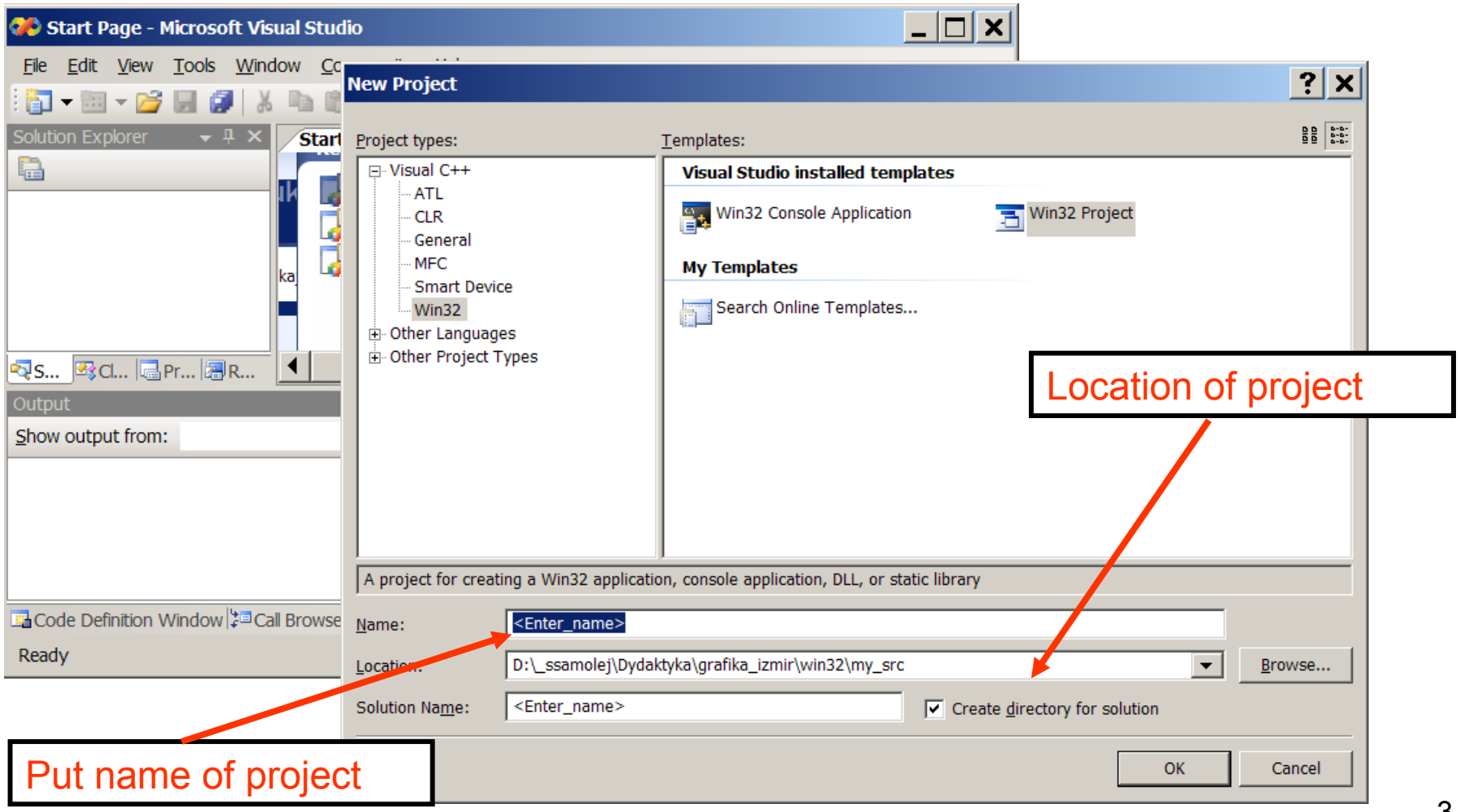
- WE WIIL BE USING WINDOWS API

Windows Programming – basic steps

- Create a new folder for project files. Prepare project.
- Create resources.
- Create source code managing resources.
- Compile, link and run application.

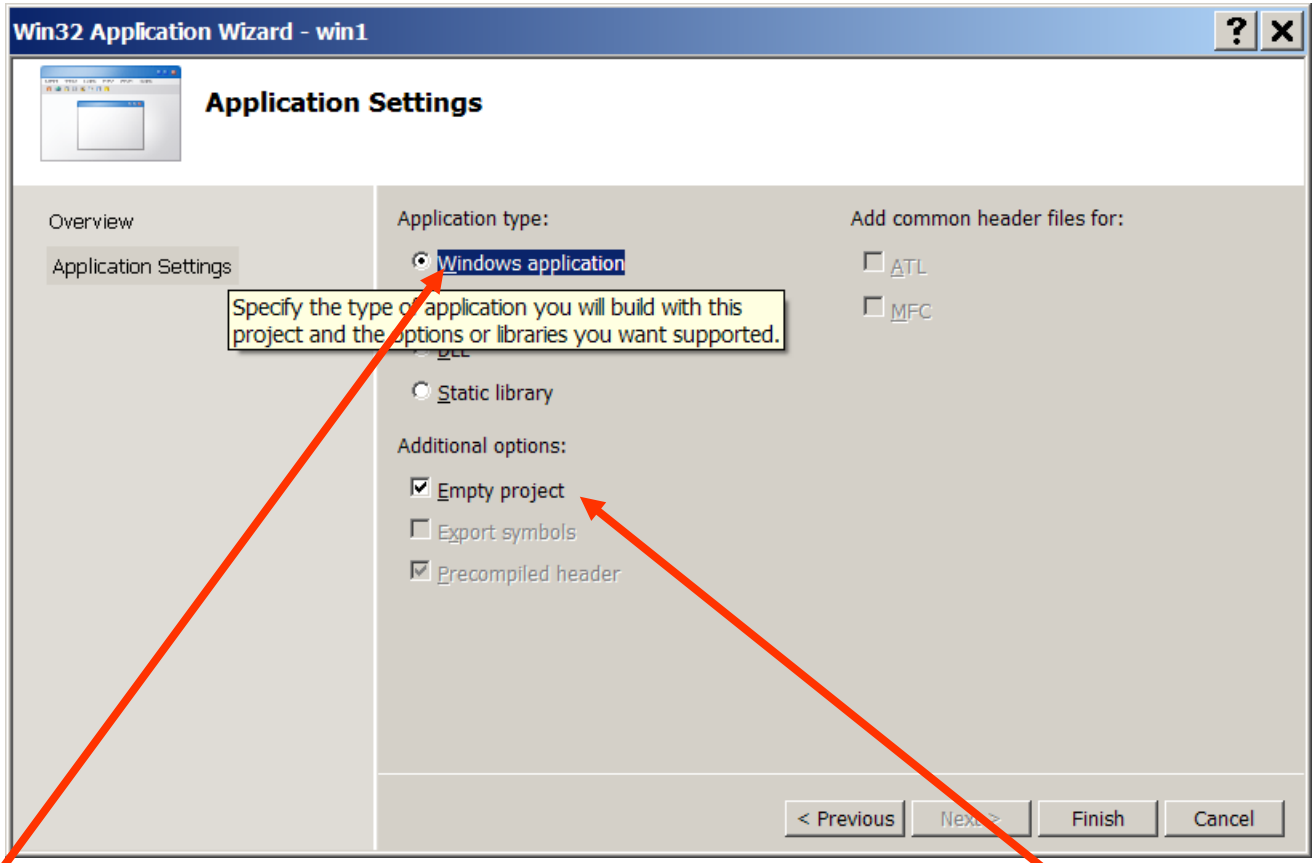
Windows Prog.– project creation (1)

- File -> New -> Project
- Choose Win32 Project



Windows Prog.– project creation (2)

- Press <<Next>> in Wizard
- Choose Empty Project Windows Application
- Press <<Finish>>

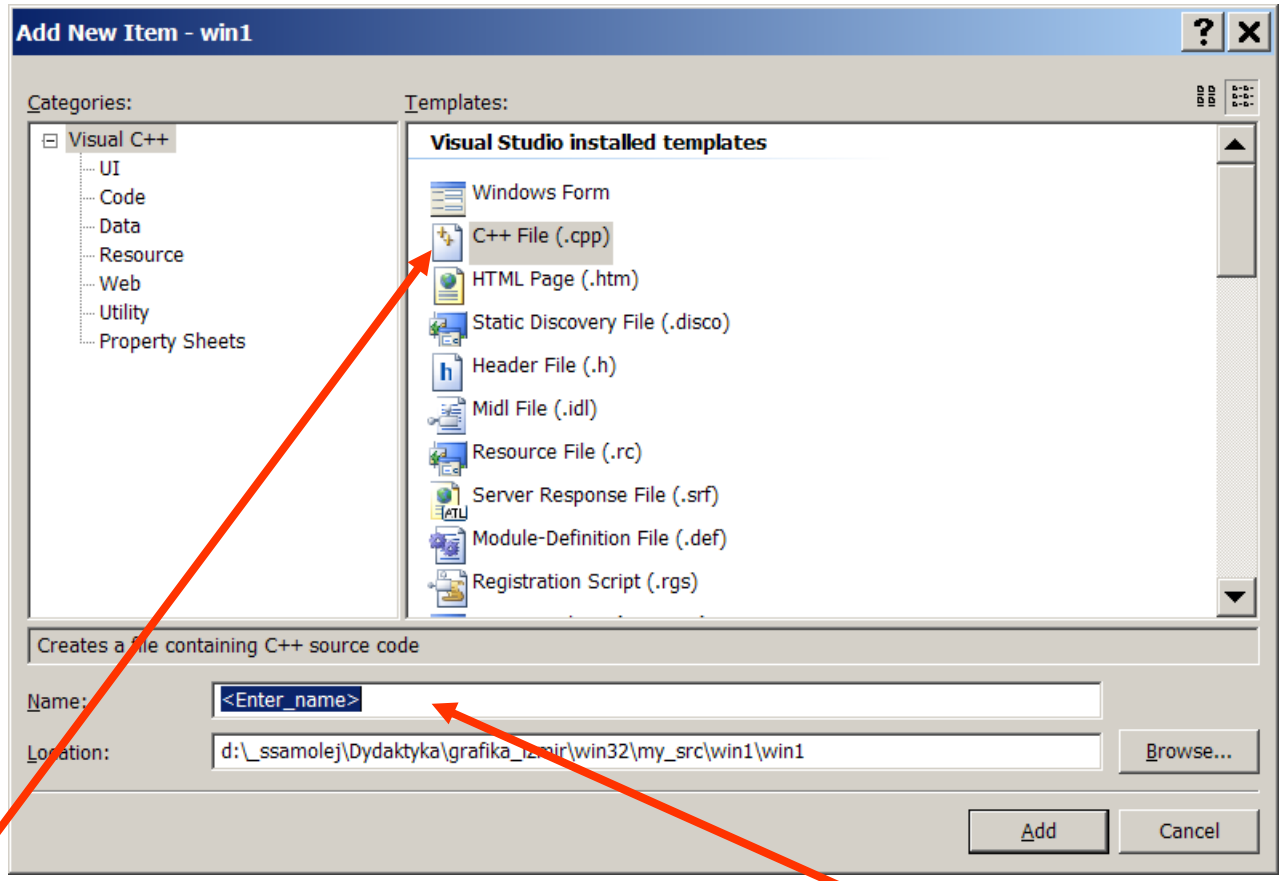


Windows Application

Empty Project

Windows Prog.– project creation (3)

- Project -> Add New Item -> C++ Source File
- Set File Name, check future location.



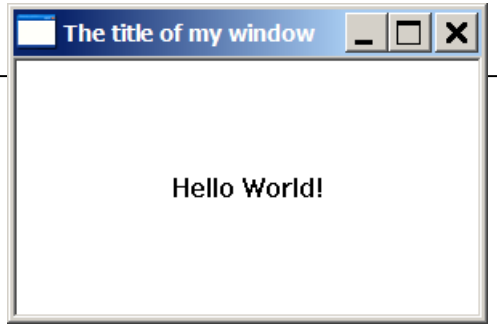
Choose C++ Source

Put File Name

A Simple Win23 Programme

```
#include <windows.h>

const char g_szClassName[] = "myWindowClass";

LRESULT CALLBACK WndProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam)
{
HDC HDCPaint; PAINTSTRUCT PaintStruct; RECT rect;
switch(msg)
{
case WM_CLOSE:
        DestroyWindow(hwnd); break;
case WM_DESTROY:
        PostQuitMessage(0); break;
case WM_PAINT:
        HDCPaint=BeginPaint(hwnd,&PaintStruct);
        GetClientRect(hwnd,&rect);
        DrawText(HDCPaint,"Hello World!",-1,&rect,                    DT_SINGLELINE|
        DT_CENTER|DT_VCENTER);
        EndPaint(hwnd, &PaintStruct);
break;
default:
        return DefWindowProc(hwnd, msg, wParam, lParam);
}
return 0;
}
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
        LPSTR lpCmdLine, int nCmdShow)
{
WNDCLASSEX wc;
HWND hwnd;
MSG Msg;

//Step 1: Registering the Window Class
wc.cbSize        = sizeof(WNDCLASSEX);
wc.style         = CS_HREDRAW | CS_VREDRAW;
wc.lpfnWndProc   = WndProc;
wc.cbClsExtra    = 0;
wc.cbWndExtra    = 0;
wc.hInstance     = hInstance;
wc.hIcon         = LoadIcon(NULL, IDI_APPLICATION);
wc.hCursor       = LoadCursor(NULL, IDC_ARROW);
wc.hbrBackground = (HBRUSH)(COLOR_WINDOW+1);
wc.lpszMenuName  = NULL;
wc.lpszClassName = g_szClassName;
wc.hIconSm                    = LoadIcon(NULL, IDI_APPLICATION);
```

```
if(!RegisterClassEx(&wc))
{MessageBox(NULL, "Window Registration Failed!", "Error!",
        MB_ICONEXCLAMATION | MB_OK);
        return 0;
}

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
        LPSTR
// Step 2: Creating the Window
hwnd = CreateWindowEx(
        WS_EX_CLIENTEDGE,
        g_szClassName,
        "The title of my window",
        WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT, CW_USEDEFAULT, 240, 120,
        NULL, NULL, hInstance, NULL);

if(hwnd == NULL)
{
        MessageBox(NULL, "Window Creation Failed!", "Error!",
                MB_ICONEXCLAMATION | MB_OK);
        return 0;
}
ShowWindow(hwnd, nCmdShow);
UpdateWindow(hwnd);

// Step 3: The Message Loop
while(GetMessage(&Msg, NULL, 0, 0) > 0)
{
        TranslateMessage(&Msg);
        DispatchMessage(&Msg);
}
return Msg.wParam;
}
```



A Simple Win32 Programm Creation Steps

- 4 Steps:
  - Registering the Window Class
  - Creating the Window
  - The Message Loop
  - The Window Procedure.

Registering the Window Class

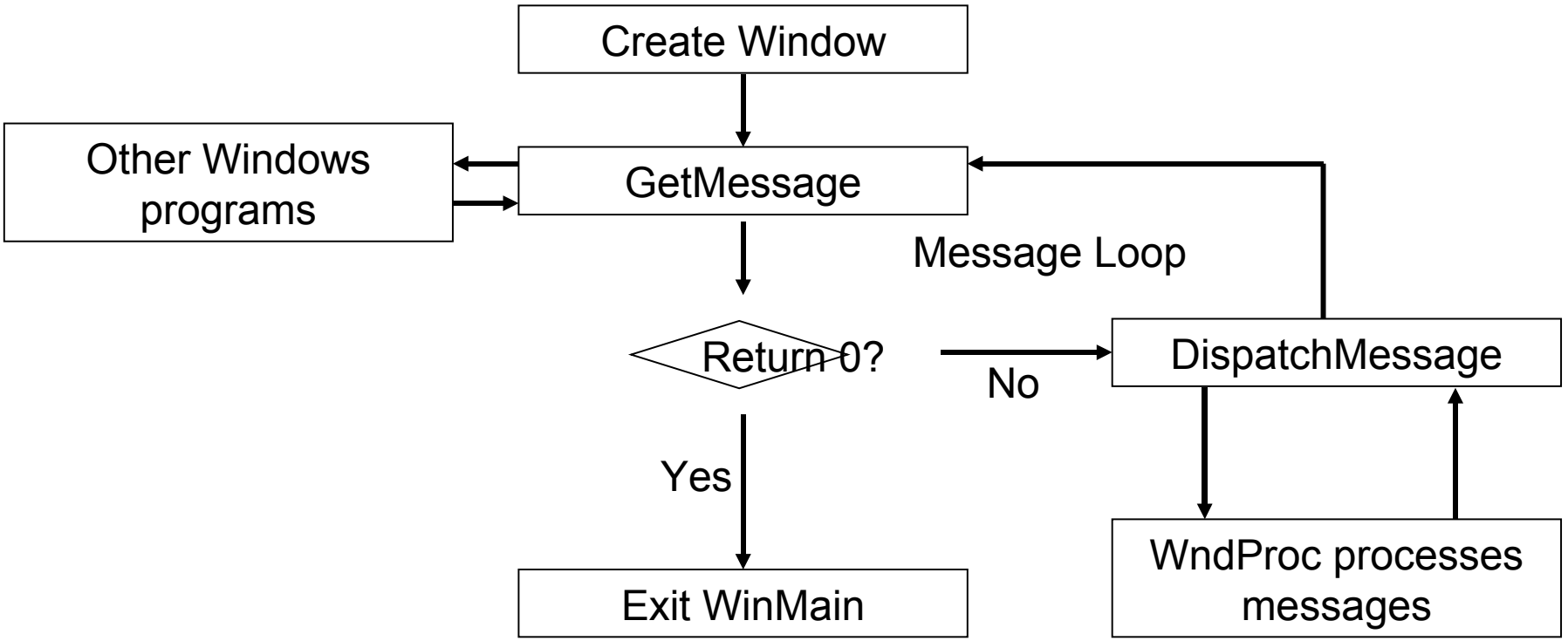
```
const char g_szClassName[] = "myWindowClass";
WNDCLASSEX wc;

//Step 1: Registering the Window Class
wc.cbSize                 = sizeof(WNDCLASSEX); //size of the structure
wc.style                  = CS_HREDRAW | CS_VREDRAW; //class styles
wc.lpfnWndProc            = WndProc; //Pointer to the window procedure for this window class
wc.cbClsExtra             = 0; // extra data allocated for class
wc.cbWndExtra             = 0; // extra data allocated for window
wc.hInstance              = hInstance; // handle to application instance
wc.hIcon                  = LoadIcon(NULL, IDI_APPLICATION); //The big icon
wc.hCursor                = LoadCursor(NULL, IDC_ARROW);  // Coursor
wc.hbrBackground          = (HBRUSH)(COLOR_WINDOW+1); // Background Colour
wc.lpszMenuName           = NULL; // Name of a menu resource
wc.lpszClassName          = g_szClassName; // Name to identify the class with
wc.hIconSm                = LoadIcon(NULL, IDI_APPLICATION); // The small icon
// The class defined must be registered in the system:
if(!RegisterClassEx(&wc))
{
        MessageBox(NULL, "Window Registration Failed!", "Error!",
                MB_ICONEXCLAMATION | MB_OK);
        return 0;
}
```

Creating the Window

```
// Step 2: Creating the Window
HWND hwnd;

hwnd = CreateWindowEx(
	WS_EX_CLIENTEDGE, // set extended window style
	g_szClassName, // use registeret windows class
	"The title of my window", // Text on window's title bar
	WS_OVERLAPPEDWINDOW, //set window style
	CW_USEDEFAULT, CW_USEDEFAULT, 240, 120, // location and window size
	NULL, NULL, hInstance, NULL);
	// parent handle, child ID, handle to instance, additional window data
// Check whether window has been created:
if(hwnd == NULL)
{
	MessageBox(NULL, "Window Creation Failed!", "Error!",
		MB_ICONEXCLAMATION | MB_OK);
	return 0;
}

ShowWindow(hwnd, nCmdShow); //paint window border, tiltle, menu…
UpdateWindow(hwnd); // paint window working area
```

The Message Loop

```
// Step 3: The Message Loop

while(GetMessage(&Msg, NULL, 0, 0) > 0) // get message from application's queue
{
        TranslateMessage(&Msg); //translate nkeyboard messages
        DispatchMessage(&Msg); // send the message to the appropriate window
}
return Msg.wParam;
```



The Window Procedure

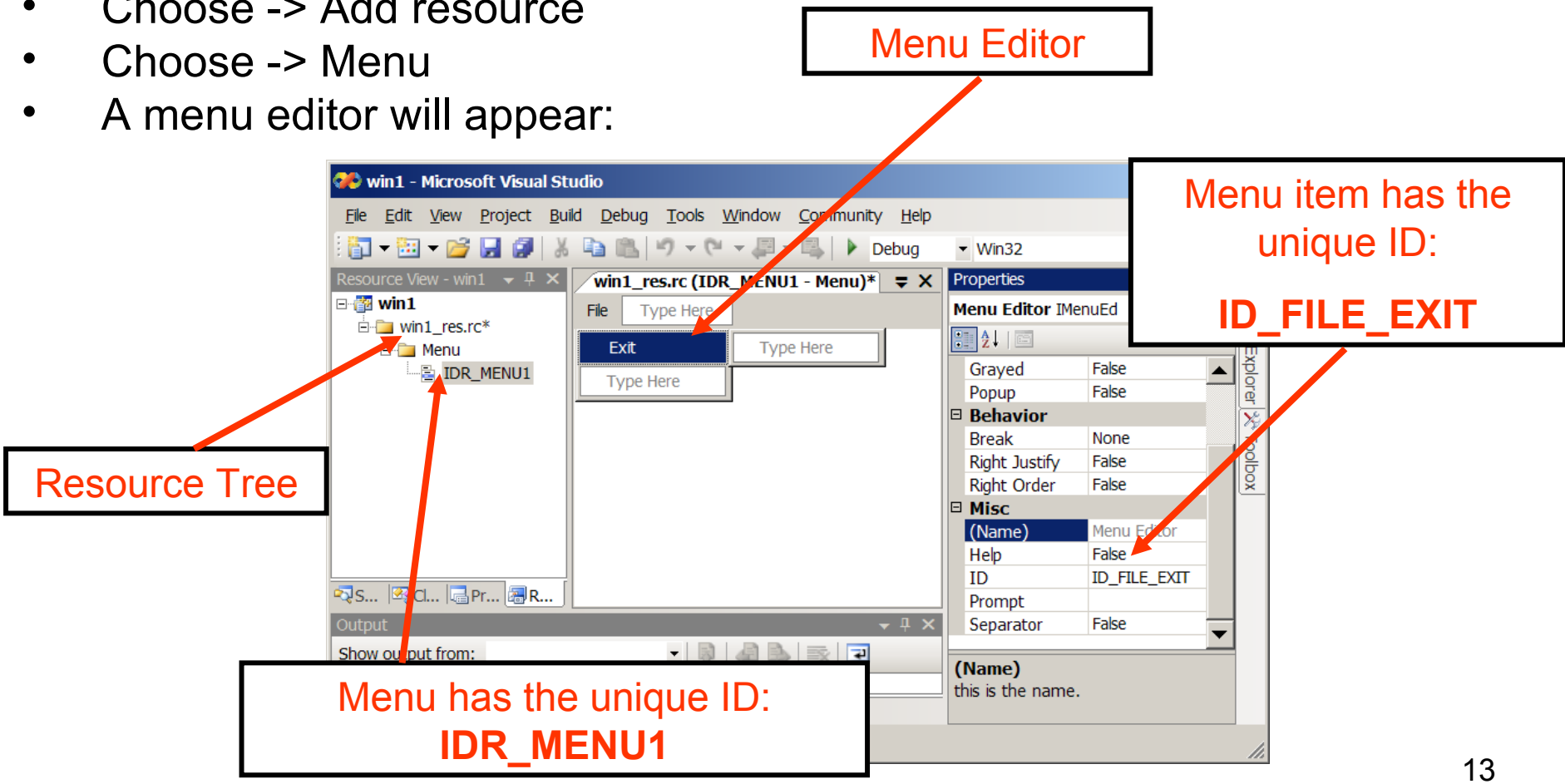
```
// Step 4: the Window Procedure
LRESULT CALLBACK WndProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM
        lParam)
{ HDC HDCPaint; PAINTSTRUCT PaintStruct; RECT rect;
switch(msg)
{
        case WM_CLOSE:
                DestroyWindow(hwnd);            break;
        case WM_DESTROY:
                PostQuitMessage(0);             break;
        case WM_PAINT:
                HDCPaint=BeginPaint(hwnd,&PaintStruct);
                GetClientRect(hwnd,&rect);
                DrawText(HDCPaint,"Hello World!",-1,&rect,
                                        DT_SINGLELINE|DT_CENTER|DT_VCENTER);
                EndPaint(hwnd, &PaintStruct);  break;
        default:
                return DefWindowProc(hwnd, msg, wParam, lParam);
        }
        return 0;
}
}
```

The Window Procedure - typical commands

```
LRESULT CALLBACK WndProc(WND hwnd, UINT iMsg,
                         WPARAM wParam, LPARAM lParam)
{       switch(iMsg)
        {
                case WM_CREATE:
                        // first message ever serviced
                case WM_SIZE:
                        // the size of window has been changed
                case WM_COMMAND:
                        // a menu item has been chosen
                case WM_CLOSE:
                        // SB. wants to close the window
                case WM_DESTROY:
                        // clean before destroying application
                case WM_PAINT:
                        // the window needs repainting
                case WM_CHAR:
                        // the letter has been send to application
                case WM_KEYDOWN: // A key has been pressed
                default:
                        return DefWindowProc(hwnd,iMsg,wParam,lParam);
        }
}
```

Adding Menu (1)

- Project -> Add New Item… -> Resource File
- A created resource file will appear in the project files tree
- Open resource tree in the project
- Right click on it
- Choose -> Add resource
- Choose -> Menu
- A menu editor will appear:



Adding Menu (2)

- Modify Window Class:

```
#include ”resource.h”
// …
const char g_szClassName[] = "myWindowClass";
WNDCLASSEX wc;
wc.cbSize                   = sizeof(WNDCLASSEX); //size of the structure
wc.style                    = CS_HREDRAW | CS_VREDRAW; //class styles
wc.lpfnWndProc      = WndProc; //Pointer to the window procedure for this window class
wc.cbClsExtra       = 0; // extra data allocated for class
wc.cbWndExtra       = 0; // extra data allocated for window
wc.hInstance        = hInstance; // handle to application instance
wc.hIcon            = LoadIcon(NULL, IDI_APPLICATION); //The big icon
wc.hCursor          = LoadCursor(NULL, IDC_ARROW);  // Coursor
wc.hbrBackground    = (HBRUSH)(COLOR_WINDOW+1); // Background Colour
wc.lpszMenuName     = MAKEINTRESOURCE(IDR_MENU1); // Name of a menu resource
wc.lpszClassName    = g_szClassName; // Name to identify the class with
wc.hIconSm          = LoadIcon(NULL, IDI_APPLICATION); // The small icon
// The class defined must be registered in the system:
if(!RegisterClassEx(&wc))
{
        MessageBox(NULL, "Window Registration Failed!", "Error!",
                MB_ICONEXCLAMATION | MB_OK);
        return 0;}
```

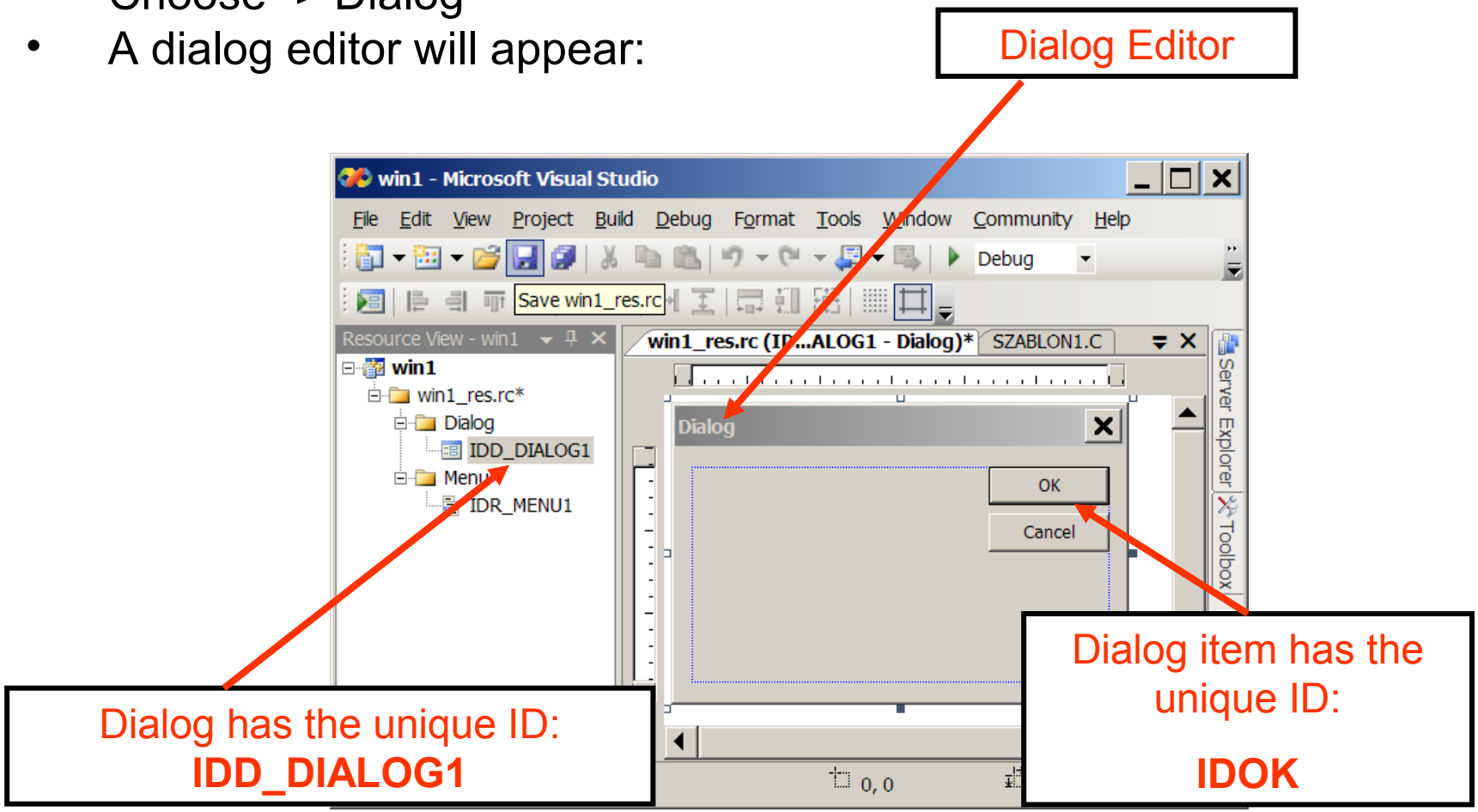
Adding Menu (3)

- Modify the Window Procedure:

```
LRESULT CALLBACK WndProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM
lParam)
{ HDC HDCPaint; PAINTSTRUCT PaintStruct; RECT rect;
switch(msg)
{           case WM_CLOSE:
                        //…
            case WM_DESTROY:
                        //…
            case WM_PAINT:
                        //…
            case WM_COMMAND:
                        switch(LOWORD(wParam))
                        {
                        case ID_FILE_EXIT:
                                    DestroyWindow(hwnd); break;
                        }
                        break;
            default:
                        return DefWindowProc(hwnd, msg, wParam, lParam);
            }
            return 0;
}}
```

Adding Dialog (1)

- Open resource tree in the project
- Right click on it
- Choose -> Add resource
- Choose -> Dialog
- A dialog editor will appear:



Adding Dialog (2)

- A new menu item may be added: File -> Dialog
- The Window Procedure sholuld be updated:

```
BOOL CALLBACK AboutDlgProc(HWND, UINT, WPARAM, LPARAM);

LRESULT CALLBACK WndProc
                        (HWND hwnd, UINT iMsg, WPARAM wParam,LPARAM lParam)
{//...
static HINSTANCE hInstance; // The handle to Instance should be stored
switch(iMsg)
{           case WM_CREATE:
                        hInstance = ((LPCREATESTRUCT) lParam) -> hInstance;
                        return 0;
            case WM_COMMAND:
                        switch(LOWORD(wParam))
                        {case ID_PLIK_KONIEC:DestroyWindow(hwnd); return 0;
                         case ID_PLIK_DIALOG: // The Dialog Proccedure should be called
                                    DialogBox(hInstance,
                                                MAKEINTRESOURCE(IDD_DIALOG1),
                                                hwnd, AboutDlgProc); return 0;
                        }return 0;
//... }
```

Adding Dialog (3)

- The Dialog Procedure sholuld be created:

```
BOOL CALLBACK AboutDlgProc (HWND hDlg, UINT message,
                                    WPARAM wParam, LPARAM lParam)
{
        switch(message)
        {
        case WM_INITDIALOG: // Assures that Dialog appears
                return TRUE;
        case WM_COMMAND:    // Control buttons service
                switch(LOWORD (wParam))
                {
                case IDOK:
                case IDCANCEL:
                        EndDialog(hDlg,0);
                        return TRUE;
                } break;
        }
        return FALSE;
}
```