

# WrapperDLL Tutorial

(by Sven-Bertil "Sweenie" Blom)

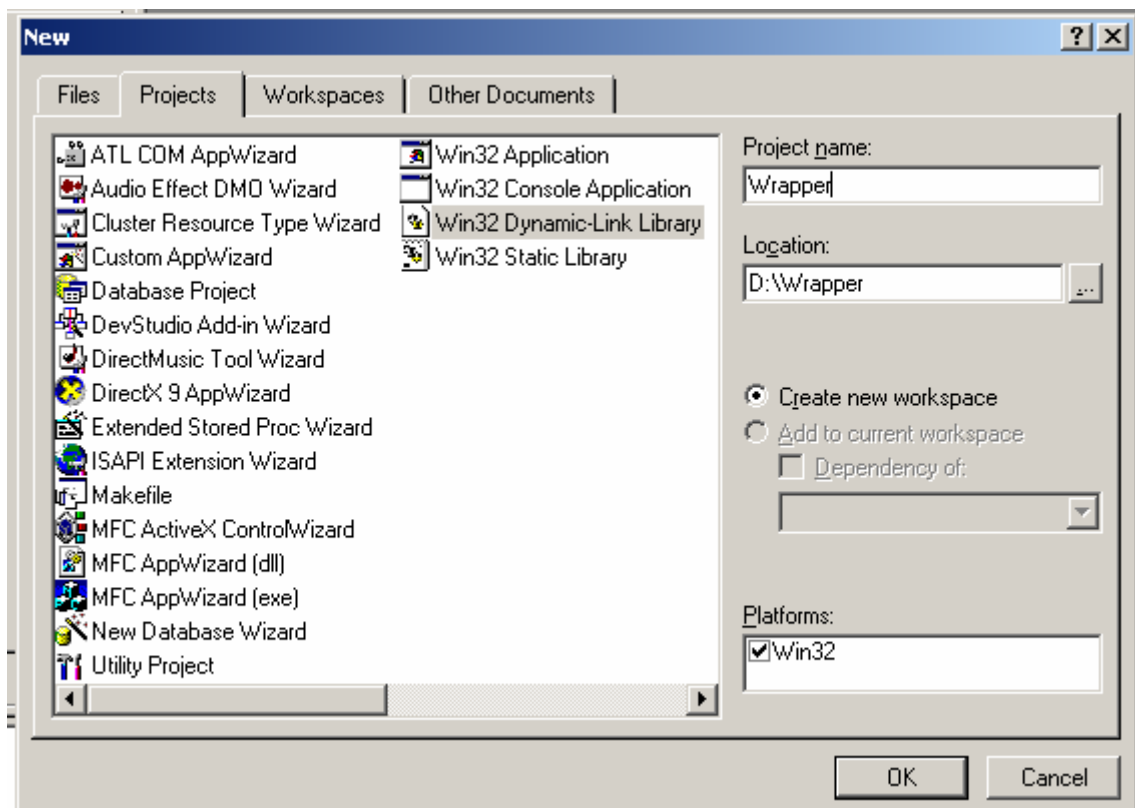
In this tutorial we will wrap some of the classes in DirectX 9 and use these to render a teapot inside of the Blitz window.

I will be using Visual C++ 6.0 but Bloodshed Dev-C++ should work fine too.  
Make sure that you have installed the DirectX 9 SDK first.

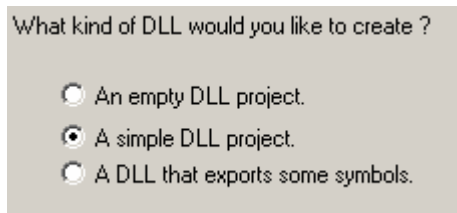
Start by opening Visual C++ and create a new project.

Select **Win32 Dynamic-Link Library** and name the project. (In this case I named it **Wrapper**).

Then press OK.



In the next step choose **A simple DLL project**.



A couple of source and headerfiles will now be created for you.

The one we are interested in is the main sourcefile. (In this case Wrapper.cpp)

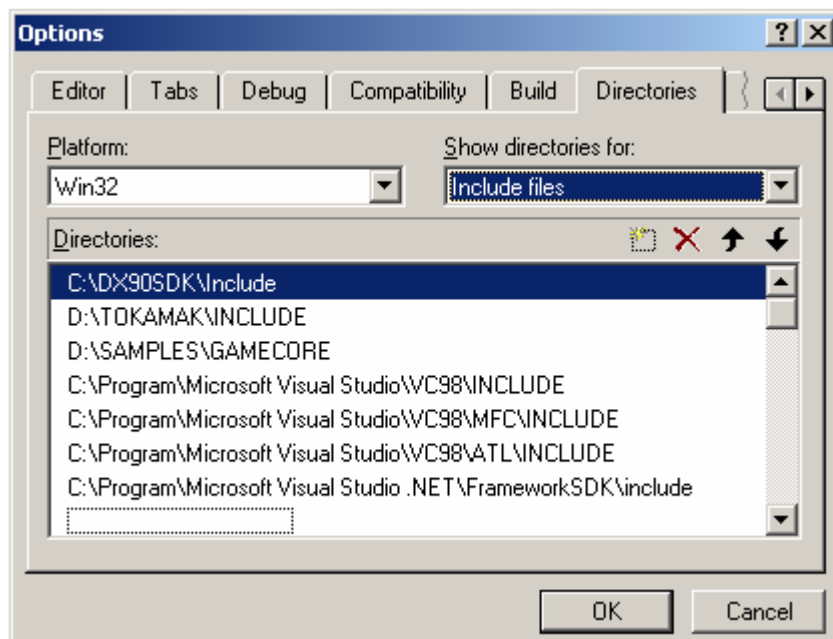
Open the Wrapper.cpp file by switching to the Fileview tab and under the Source folder doubleclick wrapper.cpp.

The sourcefile should now contain this text.

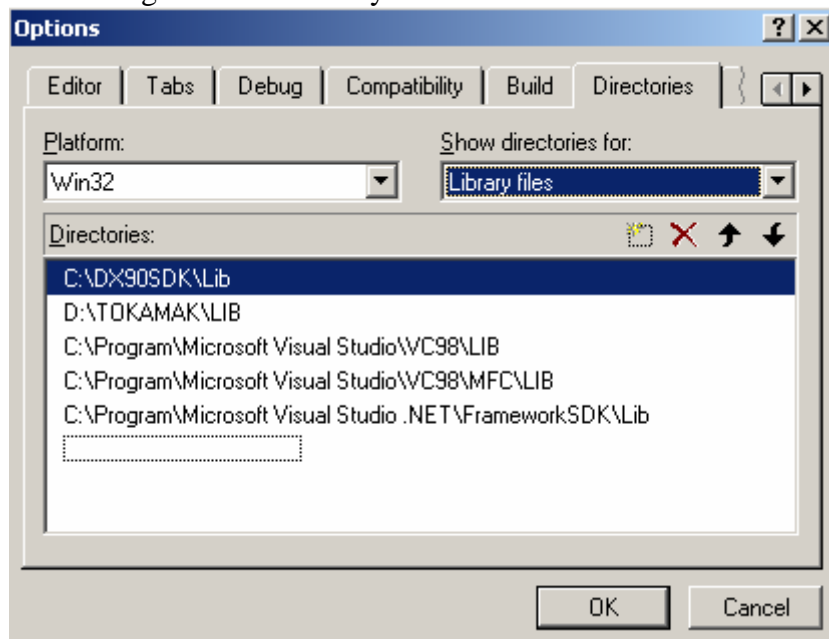
```
// wrapper.cpp : Defines the entry point for the DLL application.
//
#include "stdafx.h"

BOOL APIENTRY DllMain( HANDLE hModule,
                      DWORD  ul_reason_for_call,
                      LPVOID lpReserved
                      )
{
    return TRUE;
}
```

First we got to make sure that the compiler is aware of the DX9 include and library files. Go to the menu Tools and select Options. Select the Directories tab and make sure that the path to the DX9 include files are added to list.



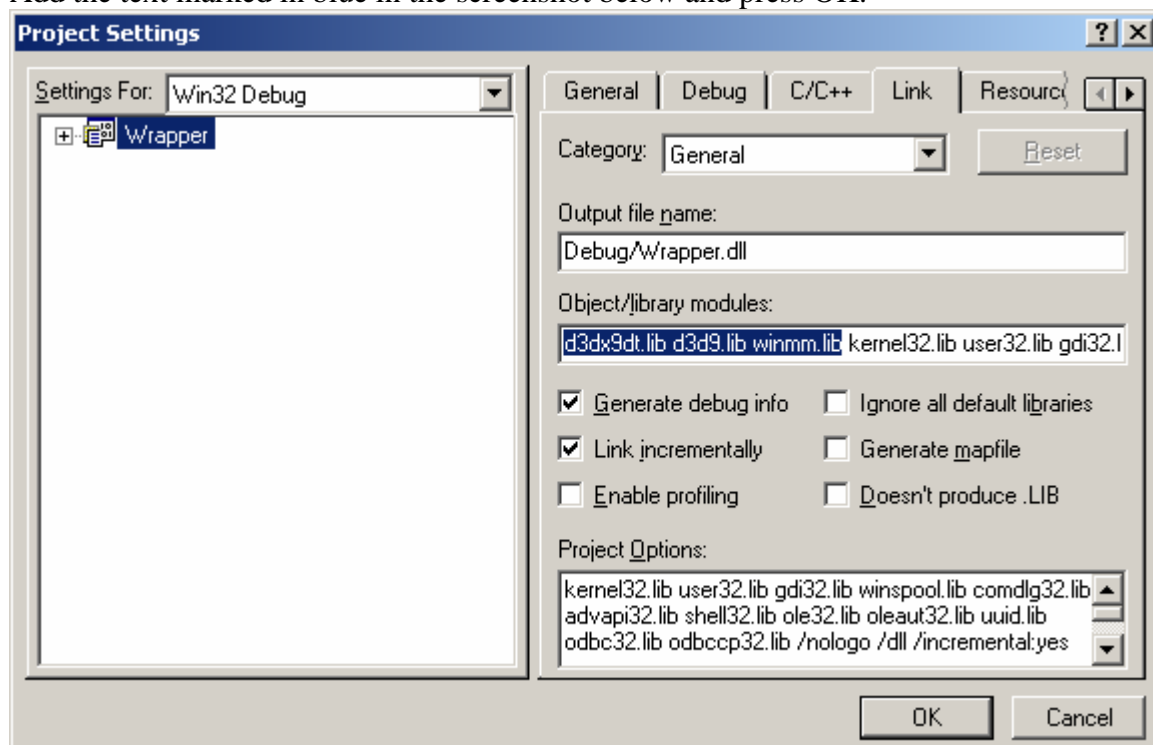
The same goes for the library files.



Press OK when you are done.

Now we must make sure that the linker includes some of the .lib files we are gonna use so go to the menu Project and select Settings (or press Alt+F7).

Add the text marked in blue in the screenshot below and press OK.



Now it's time to start coding.

We start by adding two include files.

```
#include "mmsystem.h"  
#include <d3dx9.h>
```

The first one lets us use the function `timeGetTime()` which we will use later on to rotate the teapot smoothly.

The second one includes the headerfiles for the DirectX 9 utility library. (The Direct3D headerfiles are included in this as well)

Next we define a macro which simplifies exporting a bit.

```
#define EXPORT extern "C" __declspec(dllexport)
```

Now when we want to export a function we just add the `EXPORT` macro in front and the function will be exported.

The extern "C" part will make the exported function to have this format:

`_Functionname@Sizeofparameters`

Example:

```
long MyFunction(long Param1, float Param2)
```

To export this and make it available to Blitz we write it like this instead.

```
EXPORT long __stdcall MyFunction(long Param1, float Param2)
```

The `__stdcall` makes sure we use the Standard Calling Convention which is necessary if we want the function to work with Blitz.

The exported function will now look like this:

`_MyFunction@8`

Datatype long (4 bytes) + datatype float (4 bytes) = 8 bytes

Next we define some global variables to hold our Direct3D instance, renderingdevice and teapot.

```
LPDIRECT3D9      D3D      = NULL; // The Direct3D instance  
LPDIRECT3DDEVICE9 D3DDevice = NULL; // The rendering device  
LPD3DXMESH       Teapot    = NULL; // The Teapot meshobject
```

Now it's time to create the first function.

This function will create our Direct3D instance, the renderingdevice and the teapot.

It will also create a basic light to illuminate the teapot.

I will not go into detail on this piece of code since there are plenty of tutorials on this.

```

EXPORT long __stdcall DX_InitD3D( HWND hwnd )
{
    if( NULL == ( D3D = Direct3DCreate9( D3D_SDK_VERSION ) ) )
        return E_FAIL;

    D3DPRESENT_PARAMETERS d3dpp;
    ZeroMemory( &d3dpp, sizeof(d3dpp) );
    d3dpp.Windowed = TRUE;
    d3dpp.SwapEffect = D3DSWAPEFFECT_DISCARD;
    d3dpp.BackBufferFormat = D3DFMT_UNKNOWN;
    d3dpp.EnableAutoDepthStencil = TRUE;
    d3dpp.AutoDepthStencilFormat = D3DFMT_D16;
    d3dpp.PresentationInterval = D3DPRESENT_INTERVAL_IMMEDIATE;

    if( FAILED( D3D->CreateDevice( D3DADAPTER_DEFAULT, D3DDEVTYPE_HAL, hwnd,
                                  D3DCREATE_HARDWARE_VERTEXPROCESSING,
                                  &d3dpp, &D3dDevice ) ) )
    {
        if( FAILED( D3D->CreateDevice( D3DADAPTER_DEFAULT, D3DDEVTYPE_HAL, hwnd,
                                      D3DCREATE_SOFTWARE_VERTEXPROCESSING,
                                      &d3dpp, &D3dDevice ) ) )
        {
            return E_FAIL;
        }
    }

    // Enable the Zbuffer
    D3dDevice->SetRenderState( D3DRS_ZENABLE, TRUE );

    // Create the teapot mesh
    D3DXCreateTeapot(D3dDevice,&Teapot,NULL);

    // Create a material for the teapot
    D3DMATERIAL9 mtrl;
    ZeroMemory( &mtrl, sizeof(mtrl) );
    mtrl.Diffuse.r = mtrl.Ambient.r = 1.0f;
    mtrl.Diffuse.g = mtrl.Ambient.g = 0.0f;
    mtrl.Diffuse.b = mtrl.Ambient.b = 0.0f;
    mtrl.Diffuse.a = mtrl.Ambient.a = 1.0f;
    D3dDevice->SetMaterial( &mtrl );

    // Set up a directional light
    D3DXVECTOR3 vecDir;
    D3DLIGHT9 light;
    ZeroMemory( &light, sizeof(light) );
    light.Type = D3DLIGHT_DIRECTIONAL;
    light.Diffuse.r = 1.0f;
    light.Diffuse.g = 1.0f;
    light.Diffuse.b = 1.0f;

    vecDir = D3DXVECTOR3(-1.0f,-1.0f,1.0f);
    D3DXVec3Normalize( (D3DXVECTOR3*)&light.Direction, &vecDir );

    light.Range = 1000.0f;

    D3dDevice->SetLight( 0, &light );
    D3dDevice->LightEnable( 0, TRUE);
    D3dDevice->SetRenderState( D3DRS_LIGHTING, TRUE );
    D3dDevice->SetRenderState( D3DRS_AMBIENT, 0x00202020 );

    return S_OK;
}

```

First I create the Direct3D instance.

Then I try to create the rendering device using Hardware TnL(Transform and Lighting).

If that fails, I try to create the rendering device without Hardware TnL.

If that also fails, bail out.

Then I enable the Zbuffer.

After that I create the Teapotmesh and create a material for it.

Finally I set up a directional light and some ambient light.

This function wants the handle of the window to render to.

The next function is the function that release the D3D Instance, device and the teapot when we are done.

```
EXPORT VOID __stdcall DX_Cleanup()
{
    if( Teapot != NULL )
        Teapot->Release();

    if( D3dDevice != NULL)
        D3dDevice->Release();

    if( D3D != NULL)
        D3D->Release();
}
```

The next function is the one that handles the camera and the teapot's rotation.

```
VOID SetupMatrices()
{
    D3DXMATRIXA16 matWorld;
    D3DXMatrixRotationY( &matWorld, timeGetTime()/1000.0f );
    D3dDevice->SetTransform( D3DTS_WORLD, &matWorld );

    D3DXVECTOR3 vEyePt( 0.0f, 3.0f,-5.0f );
    D3DXVECTOR3 vLookatPt( 0.0f, 0.0f, 0.0f );
    D3DXVECTOR3 vUpVec( 0.0f, 1.0f, 0.0f );
    D3DXMATRIXA16 matView;
    D3DXMatrixLookAtLH( &matView, &vEyePt, &vLookatPt, &vUpVec );
    D3dDevice->SetTransform( D3DTS_VIEW, &matView );

    D3DXMATRIXA16 matProj;
    D3DXMatrixPerspectiveFovLH( &matProj, D3DX_PI/4, 1.33f, 1.0f, 1000.0f );
    D3dDevice->SetTransform( D3DTS_PROJECTION, &matProj );
}
```

As you can see I don't export this function since I will call this function from within the renderfunction.

I could have exported this and added some parameters so that blitz could move the camera.

But since this example only demonstrates a rotating teapot I choose not to.

Now comes the actually rendering function.

```

EXPORT VOID __stdcall DX_Render()
{
    if( NULL == D3dDevice )
        return;

    // Clear the backbuffer to a blue color and empty the Zbuffer
    D3dDevice->Clear( 0, NULL, D3DCLEAR_TARGET | D3DCLEAR_ZBUFFER,
        D3DCOLOR_XRGB(0,0,255), 1.0f, 0 );

    if( SUCCEEDED( D3dDevice->BeginScene() ) )
    {
        SetupMatrices();
        Teapot->DrawSubset(0);
        D3dDevice->EndScene();
    }

    // Flip the backbuffer
    D3dDevice->Present( NULL, NULL, NULL, NULL );
}

```

First I clear the backbuffer and empty the Zbuffer.

Then I call the SetupMatrices function which set the current rotationmatrix for the Teapot and the camera.

It also sets the correct projectionvalues.

Then I render the teapot by calling the Teapotobject's method DrawSubset.

Subsets could be compared to Blitz's surfaces.

Finally I present the rendering on screen.

To allow for some interaction I added a function that lets you decide if the teapot should be rendered as wireframe or not.

```

EXPORT VOID __stdcall DX_wireframe(int Enable)
{
    if (Enable==TRUE)
        D3dDevice->SetRenderState( D3DRS_FILLMODE, D3DFILL_WIREFRAME);
    else
        D3dDevice->SetRenderState( D3DRS_FILLMODE, D3DFILL_SOLID);
}

```

Now when you compile this project the compiler will produce a file called wrapper.dll.

If you open this dll with PEvent or Dumpbin /EXPORTS you will see the following functions:

```

_DX_InitD3D@4
_DX_Cleanup@0
_DX_Render@0
_DX_Wireframe@4

```

Using these it's time to create the userlib declarationfile.

First create a empty textfile called Wrapper.decls  
Then add the following text:

```
.lib "wrapper.dll"

DX_InitD3D%(Hwnd%): "_DX_InitD3D@4"
DX_Cleanup(): "_DX_Cleanup@0"
DX_Render(): "_DX_Render@0"
DX_Wireframe(Enable%): "_DX_Wireframe@4"
```

Save it in the userlibs directory in the Blitzfolder.

Since DX\_InitD3D want the Blitz window handle you may need to create a second decls file to add some functions from USER32.DLL to retrieve this window handle.

Create a second decls file called user32.decls or something like that.  
Add the following text to it.

```
.lib "user32.dll"

GetActivewindow%(): "GetActivewindow"
```

Put it in the same place as the wrapper.decls file.

Now copy the compiled dll to the userlibs directory and create a new blitzproject.

Enter this piece of code and run the program:

```
Graphics 800,600,0,2
hwnd%=GetActivewindow()

result%=DX_InitD3D(hwnd)
If result<>0 Then
    Print "Couldn't initialize Direct3D!"
    Waitkey
End
EndIf

While Not KeyHit(1)
    If KeyHit(57) Then
        Switch=1-Switch
        If Switch=0 Then DX_Wireframe(False)
        If Switch=1 Then DX_Wireframe(True)
    EndIf

    DX_Render
Wend

DX_Cleanup

End
```



If everything was done correct you should see a rotating red teapot within the Blitz window.  
That's about it.

I would like to mention that this DLL could be used with visual basic or any other language that supports API-calls as well.

//Sven-Bertil Blom

Last modified: 12 Sep. 2004