Speeding Up Your DirectX8 Functions In Visual Basic Tutorial

Contents

Intorduction

I decided to make this tutorial and some demo’s because I have had a lot of game developers asking me how to make and reference .dll’s made in C++ to speed up their game engines. This will help not only game developers but it will also help any programmer wanting the knowledge on how to reference .dll’s from C++. If your wanting to make a good engine in Visual Basic then this is for you. This tutorial will show you step-by-step on how to make the .dll’s and reference them. Once you learn the logic behind it, it will be simple to you and you will be able to improve your engine with no problem.

Basic Visual Basic Optimizing Tips

These are just some basic optimizing tips for Visual Basic that should be implemented in every single application. The reason this section is here is because I see this problem all the time, not only in business applications but also in multimedia applications.

1) Always use "Option Explicit."
   -This will force you to define every variable. See #2 for further explanation.

2) Always define your variables.
   -If this is not done the Visual Basic's compiler will have to decide what kind of variable you are trying to use then convert them to their correct data types.

   Examples:
   -Never define your variables like this.
   ```
   Dim I, j, x, z As Long
   ```
   This means only z is defined as long.

   -Instead define your variable as follows.
   ```
   Dim I As Long, j As Long, x As Long, z As Long
   ```
   Now every variable is defined as long.

3) Precompute Complicated Math.
   -If you know your going to use the same math function over and over or it's a complicated math problem precompute it. If you

don't it will be computed every single time you use it. Which makes the function slower than it should be.

Example:
 This is a bad example but it shows what I mean.

```
   Private Function Whatever(Time As Single) As Long
    Dim I As Long

    For I = 0 To 1000
     Time = (Time * 0.5) * Sin(PI * 3) / 2
     If Time > 500 Then
      Time = Time / 2
     End If
     Whatever = cSng(Time)
    Next
   End Function
```

Instead make the function as follows.

```
 Private Function Whatever(Time As Single) As Long
    Dim I As Long
    Dim t As Long

    t = (Time * 0.5) * Sin(PI * 3) / 2
    For I = 0 To 1000
     If t > 500 Then
      t = t / 2
     End If
     Whatever = cSng(t)
    Next

 End Function
```

4) Simplify The Math
   -Never use the operator function "^". Always write it out. This will speed up your application a lot. In some cases the "^" took 8.457 seconds while the multiplication took 0.517 seconds. Major improvement.

   Example:

```
Test = (X ^ 2) + (Y ^ 2) + (Z ^ 2)
```

   Instead write it like this.

```
Test = (X * X) + (Y * Y) + (Z * Z)
```

Setting Up C++

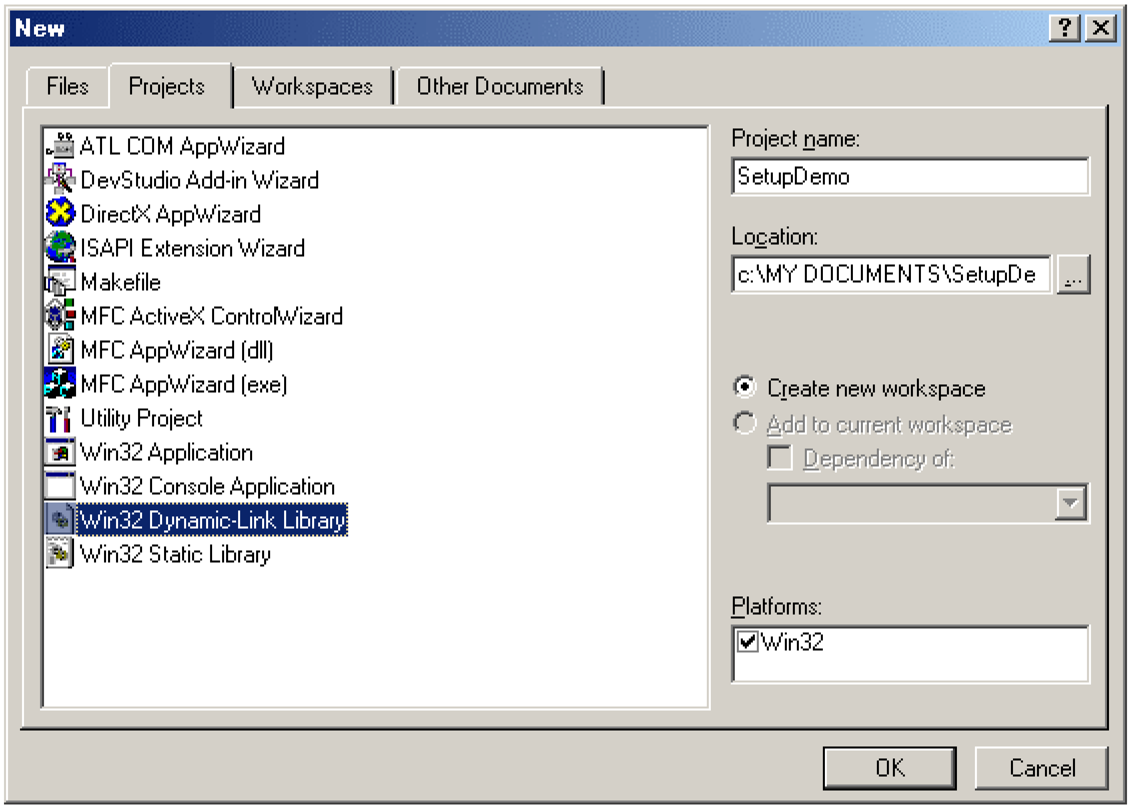
To setup C++ to use in Visual Basic is simple once you know how to do it. Here I will show you how to set it up step-by-step.

Step 1: Creating The Project

Startup Visual C++ 6 and choose File->New.

Step 2: Setup Dialog 1

In this dialog box select "Win32 Dynamic-Link Library" then enter the name you want for your project and the location it is to be stored. See "Project Name:" and "Location:" When finished select "Ok" to proceed.



Project name: SetupDemo

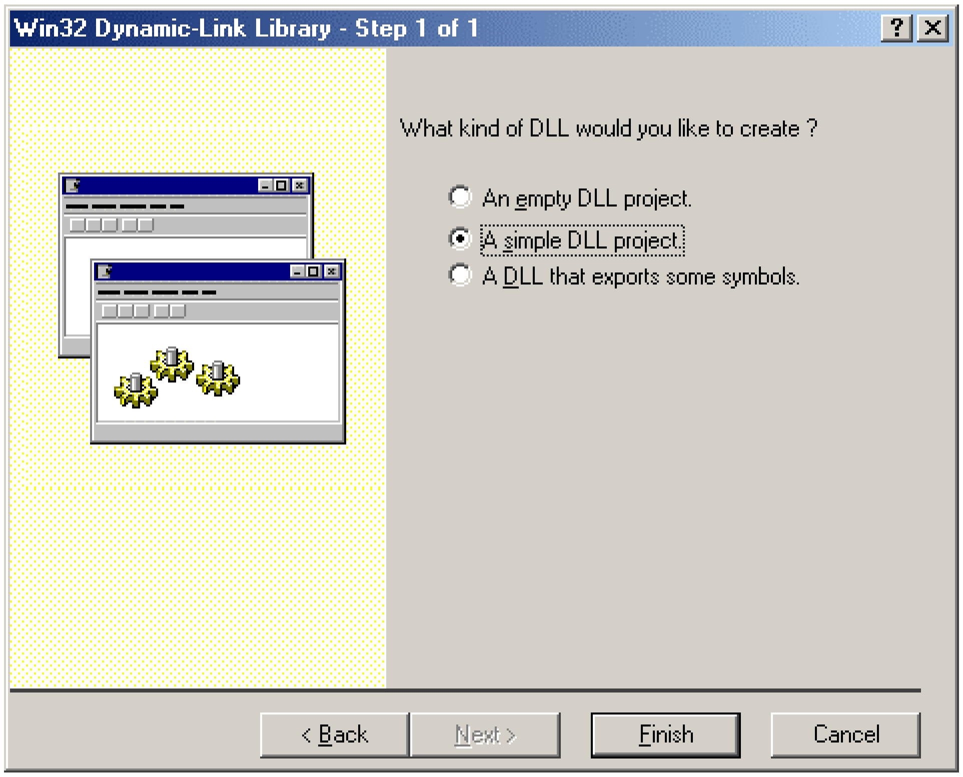
Location: c:\MY DOCUMENTS\SetupDe

[✓ Create new workspace]

Platforms: [✓ Win32]

Step 3: Setup Dialog 2

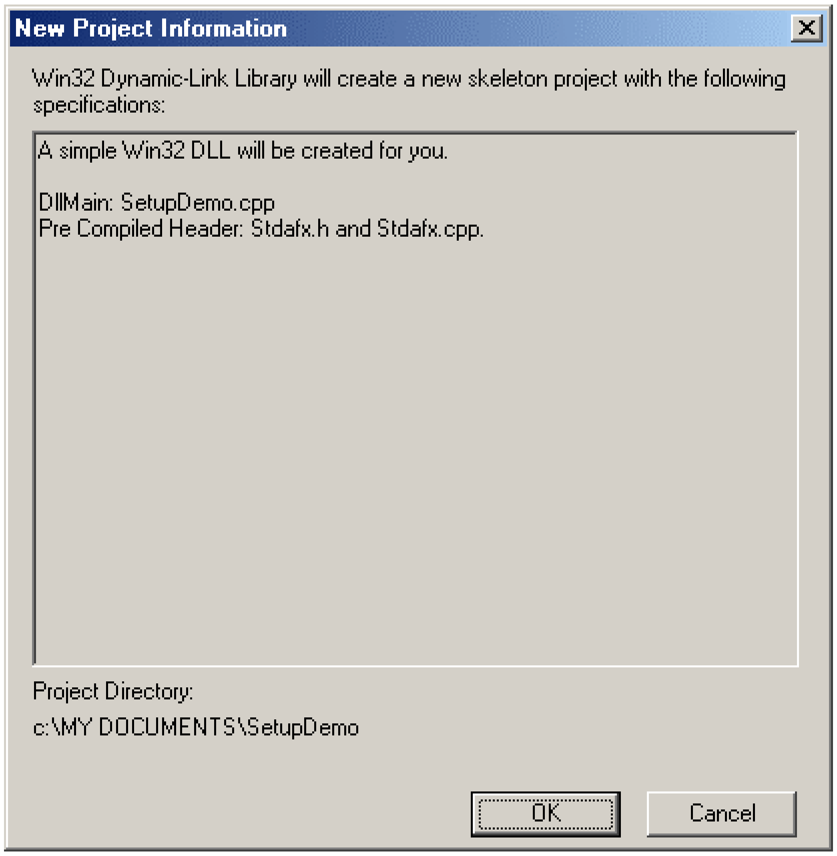
In the next dialog box select "A Simple DLL project" and hit "Finish."


Win32 Dynamic-Link Library - Step 1 of 1
What kind of DLL would you like to create ?
An empty DLL project.
[✓ A simple DLL project.]
A DLL that exports some symbols.
< Back
Next >
Finish
Cancel

Step 4: Setup Dialog 3

This setup dialog just gives you some basic information about your project. Just hit "Ok" and proceed.

See Below On Next Page.



New Project Information

Win32 Dynamic-Link Library will create a new skeleton project with the following specifications:

A simple Win32 DLL will be created for you.

DllMain: SetupDemo.cpp
Pre Compiled Header: Stdafx.h and Stdafx.cpp.

Project Directory:
c:\MY DOCUMENTS\SetupDemo

Now you have a basic setup and are ready to begin the project.

Setting Up The C++ Header File

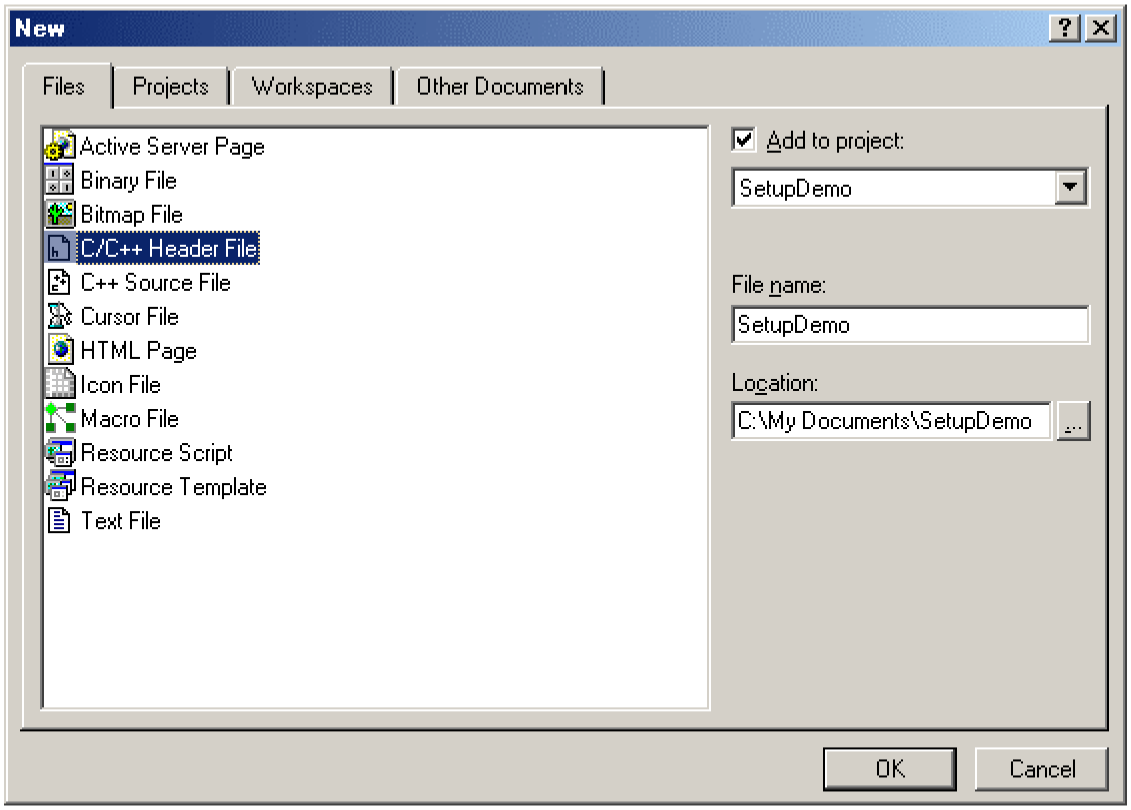
Setting up the header file for the project is not that difficult. A header file is not needed but it will help to keep the code clean.

The header file I will use consists of:

-Included header file names
-Structures –Like types in Visual Basic
-Math Functions (Vector)
-Prototypes

To create the header file choose File->New and select "C/C++ Header File." Then give the header file a name in the "File Name" text box. Then hit "Ok" to proceed.

Note: Make sure the "Add To Project" is checked and the correct project name is selected.



[✓] Add to project: SetupDemo

File name: SetupDemo

Location: C:\My Documents\SetupDemo

Starting The C++ Header File

Since we are dealing with DirectX 8 and Visual Basic theres some important things to discuss. First off C++ DirextX 8 types and Visual Basic DirectX 8 types are different. By different I mean the names do not match. For instance in C++ the Vector3 type is D3DXVECTOR and in Visual Basic its D3DVECTOR. So we need to create our own to fit Visual Basics needs.

Keep This In Mind:

C++		Visual Basic
Float	=	Single
Long	=	Long
Bool	=	Boolean

With that in mind lets start the header file.

First we need to include the math header in able to use the Sqrt() function.

```
#include <Math.h>
```

Then we define our data structures to use with Visual Basic. I'm assume you know what these do. Same as Visual Basic but in C++.

Sample:

```
typedef struct D3DVECTOR2
{
     float X;
     float Y;
} _D3DVECTOR2;
```

Same as saying:

```
Private Type D3DVECTOR2
     X As Single
     Y As Single
End Type
```

Now heres the types definitions we need to declare.(I added a few extra so you get the idea better!)

```
typedef struct D3DVECTOR2
{
     float X;
     float Y;
} _D3DVECTOR2;

typedef struct D3DVECTOR
{
     float X;
     float Y;
     float Z;
} _D3DVECTOR;

typedef struct D3DVECTOR4
{
     float X;
     float Y;
     float Z;
     float W;
} _D3DVECTOR4;

typedef struct D3DVERTEX
{
     float X;
     float Y;
     float Z;
     float nX;
     float nY;
     float nZ;
     float tU;
     float tV;
} _D3DVERTEX;

typedef struct D3DVERTEX2
{
     float X;
     float Y;
     float Z;
     float nX;
     float nY;
     float nZ;
     float tU1;
     float tV1;
     float tU2;
     float tV2;
} _D3DVERTEX2;
```

```
typedef struct D3DLVERTEX
{
     float X;
     float Y;
     float Z;
     float tU;
     float tV;
     long Color;
     long Specular;
} _D3DLVERTEX;

typedef struct D3DLVERTEX2
{
     float X;
     float Y;
     float Z;
     float tU1;
     float tV1;
     float tU2;
     float tV2;
     long Color;
     long Specular;
} _D3DLVERTEX2;

typedef struct D3DTLVERTEX
{
     float X;
     float Y;
     float Z;
     float tU;
     float tV;
     float RHW;
     long Color;
     long Specular;
} _D3DTLVERTEX;
```

```
typedef struct D3DTLVERTEX2
{
     float X;
     float Y;
     float Z;
     float tU1;
     float tV1;
     float tU2;
     float tV2;
     float RHW;
     long Color;
     long Specular;
} _D3DTLVERTEX2;
```

These will be the same as visual basic now.

Adding C++ Functions

C++ functions for Visual Basic (*That I Use*) are setup like this.

```
_declspec(dllexport) <Return Type> _stdcall <Function()>
{
     return <Return Type>;
}
```

If it is a void it doesn't return a value for instance:

```
_declspec(dllexport) void _stdcall <Function()>
{
     This will return nothing
}
```

Sample:

```
_declspec(dllexport) float _stdcall PI(void)
{
     return 3.141f;
}
```

In Visual Basic it would be the same as:

```
Public Function PI() As Single
     PI = 3.141
End Function
```

With this in mind lets begin adding functions.

Prototypes - Prototypes declare a function. They are also statements and every statement in C++ end with a semicolon. In the C++ file they will not end with a semicolon. It's a bit confusing but I'll explain it all.

Let's Declair a prototype.
In this tutorial I will demostrate 2 functions and you can build from there.

Our first prototype:

```
_declspec(dllexport) bool _stdcall FlipMeshNormals(D3DVERTEX *Vertices,
long NumVertices);

_declspec(dllexport) float _stdcall Distance3D(D3DVECTOR& Vector1,
D3DVECTOR& Vector2);
```

Get the idea? Pretty simple? Wait till you do it a few times.

The Complete Header File

```
//Included Header Files
#include <Math.h>

//Type Definition Structures
typedef struct D3DVECTOR2
{
	float X;
	float Y;
} _D3DVECTOR2;

typedef struct D3DVECTOR
{
	float X;
	float Y;
	float Z;
} _D3DVECTOR;

typedef struct D3DVECTOR4
{
	float X;
	float Y;
	float Z;
	float W;
} _D3DVECTOR4;

typedef struct D3DVERTEX
{
	float X;
	float Y;
	float Z;
	float nX;
	float nY;
	float nZ;
	float tU;
	float tV;
} _D3DVERTEX;
```

```
typedef struct D3DVERTEX2
{
     float X;
     float Y;
     float Z;
     float nX;
     float nY;
     float nZ;
     float tU1;
     float tV1;
     float tU2;
     float tV2;
} _D3DVERTEX2;

typedef struct D3DLVERTEX
{
     float X;
     float Y;
     float Z;
     float tU;
     float tV;
     long Color;
     long Specular;
} _D3DLVERTEX;

typedef struct D3DLVERTEX2
{
     float X;
     float Y;
     float Z;
     float tU1;
     float tV1;
     float tU2;
     float tV2;
     long Color;
     long Specular;
} _D3DLVERTEX2;
```

```
typedef struct D3DTLVERTEX
{
     float X;
     float Y;
     float Z;
     float tU;
     float tV;
     float RHW;
     long Color;
     long Specular;
} _D3DTLVERTEX;

typedef struct D3DTLVERTEX2
{
     float X;
     float Y;
     float Z;
     float tU1;
     float tV1;
     float tU2;
     float tV2;
     float RHW;
     long Color;
     long Specular;
} _D3DTLVERTEX2;


//Prototypes
_declspec(dllexport) bool _stdcall
FlipMeshNormals(D3DVERTEX *Vertices, long NumVertices);

_declspec(dllexport) float _stdcall Distance3D(D3DVECTOR&
Vector1, D3DVECTOR& Vector2);
```

Setting Up The C++ File

The C++ file I will use consists of:

- -Included header file names
- -DLL Entry
- -Functions

The C++ file should already be created. It should have the same name as your project.

The only thing that needs to be done to setup the C++ file is the included file and to change the DLL entry code.

The included files are:

```
#include "stdafx.h"         //Included automatically
#include <Math.h>           //Same as the header file
#include "SetupDemo.h"      //Include or header file
```

Then we need to change this:

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

To This: (Better)

```
bool _stdcall DllMain(HANDLE hModule, DWORD
ul_reason_for_call, LPVOID lpReserved)
{
     switch (ul_reason_for_call)
     {
          case DLL_PROCESS_ATTACH:
          case DLL_THREAD_ATTACH:
          case DLL_THREAD_DETACH:
          case DLL_PROCESS_DETACH: break;
     }

     return true;
}
```

That's it! Not to bad so far. Now lets start the actual functions!

Adding The Functions To The C++ File

Like I was talking about in the header file section, protypes use semicolons and the C++ file doesn't. Like I said the reason the prototypes use a semicolon and the C++ doesn't is because prototypes are statements. All we do is declair the same function without the semicolon.

```
_declspec(dllexport) bool _stdcall
FlipMeshNormals(D3DVERTEX *Vertices, long NumVertices)
```

See same thing! Not to bad. Now comes the fun. Writing the actual function. Lets do the first one.

Error handling:

```
_declspec(dllexport) bool _stdcall
FlipMeshNormals(D3DVERTEX *Vertices, long NumVertices)
{
     try{
          //Function goes here
          return true;
     }catch(…){
          return false;
     }
}
```

In Visual Basic it's the same as saying:

```
Public Function FlipMeshNormals() As Boolean
On Local Error GoTo ErrOut

  'Function Goes Here
  FlipMeshNormals = True
  Exit Function

ErrOut:
 FlipMeshNormals = False
End Function
```

Pretty cool huh! ☺

Now lets do the actual function.

```
_declspec(dllexport) bool _stdcall
FlipMeshNormals(D3DVERTEX *Vertices, long NumVertices)
{
     try{
          for(long i = 0; i < NumVertices; i++)
          {
               Vertices->nX = -Vertices->nX;
               Vertices->nY = -Vertices->nY;
               Vertices->nZ = -Vertices->nZ;
               Vertices++;
          }
          return true;
     }catch(…){
          return false;
     }
}
```

That's it!!! Now lets do the other function.

```
_declspec(dllexport) float _stdcall Distance3D(D3DVECTOR&
Vector1, D3DVECTOR& Vector2)
{
     float Distance;

     Distance = sqrtf((Vector2.X - Vector1.X) *
                      (Vector2.X - Vector1.X) +
                      (Vector2.Y - Vector1.Y) *
                      (Vector2.Y - Vector1.Y));
     return Distance;
}
```

AWESOME!

We could simplify that last one. We could just say.

```
_declspec(dllexport) float _stdcall Distance3D(D3DVECTOR&
Vector1, D3DVECTOR& Vector2)
{
     return sqrtf((Vector2.X - Vector1.X) *
                  (Vector2.X - Vector1.X) +
                  (Vector2.Y - Vector1.Y) *
                  (Vector2.Y - Vector1.Y));
}
```

Better Yet!

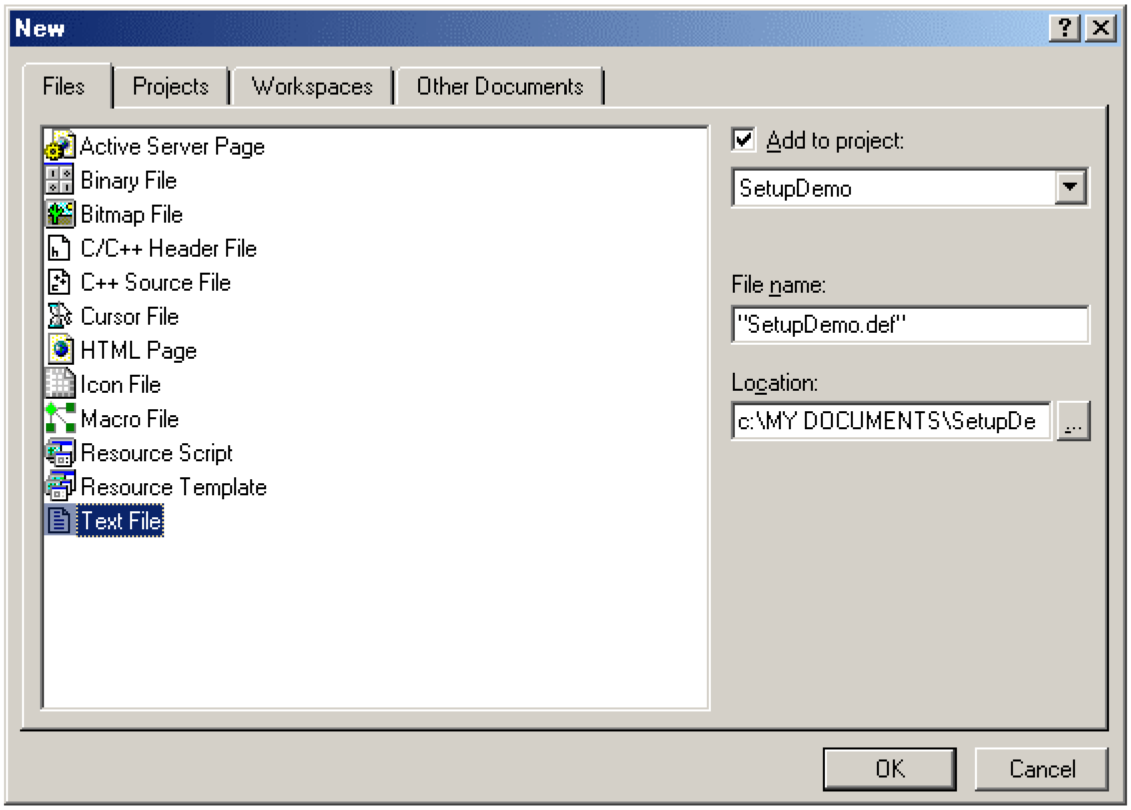
The Complete C++ File

```
#include "stdafx.h"          //Included automatically
#include <Math.h>            //Same as the header file
#include "SetupDemo.h"       //Include or header file

bool _stdcall DllMain(HANDLE hModule, DWORD
ul_reason_for_call, LPVOID lpReserved)
{
     switch (ul_reason_for_call)
     {
          case DLL_PROCESS_ATTACH:
          case DLL_THREAD_ATTACH:
          case DLL_THREAD_DETACH:
          case DLL_PROCESS_DETACH: break;
     }

     return true;
}

_declspec(dllexport) bool _stdcall
FlipMeshNormals(D3DVERTEX *Vertices, long NumVertices)
{
     try{
          for(long i = 0; i < NumVertices; i++)
          {
               Vertices->nX = -Vertices->nX;
               Vertices->nY = -Vertices->nY;
               Vertices->nZ = -Vertices->nZ;
               Vertices++;
          }
          return true;
     }catch(…){
          return false;
     }
}

_declspec(dllexport) float _stdcall Distance3D(D3DVECTOR&
Vector1, D3DVECTOR& Vector2)
{
     return sqrtf((Vector2.X - Vector1.X) *
                  (Vector2.X - Vector1.X) +
                  (Vector2.Y - Vector1.Y) *
                  (Vector2.Y - Vector1.Y));
}
```

The .def File

The .def file is where you export the functions. To create the .def file follow these simple steps. Select File->New and proceed.

Step 1 – Setup Dialog 1
Select "Text File" and for the file name make sure you have the quotes and put <"Name.def">. Hit "Ok" and Name.def will be added to the project.



☑ Add to project: SetupDemo

File name: "SetupDemo.def"

Location: c:\MY DOCUMENTS\SetupDe

In the .def file it sets up like this:

```
LIBRARY   SetupDemo ;Name of the library
EXPORTS             ;And the exports section

     ;The functions you want to export
     FlipMeshNormals @1
     Distance2D      @2
```

That's it! That's the easiest part of the .dll. ☺

Compiling The .dll

To compile simply make sure all the files are there, you have the header file in the #include section, and that’s it. It should compile fine. If you have trouble compiling let me know and I’ll help try to help.

Converting The Exported Functions

Converting the functions is pretty simple.
Be sure to reference DirectX 8.

As I showed above:

```
C++      Visual Basic
Float = Single
Long  = Long
Bool  = Boolean
```

With that in mind lets convert our functions. It will look just like an API.

C++

```
_declspec(dllexport) bool _stdcall
FlipMeshNormals(D3DVERTEX *Vertices, long NumVertices)
```

Visual Basic

```
Public Declare Function FlipMeshNormals Lib "SetupDemo.dll"
(ByRef Vertices As D3DVERTEX, ByVal NumVertices As Long) As
Boolean
```

C++

```
_declspec(dllexport) bool _stdcall Distance2D(D3DVECTOR2&
Vector1, D3DVECTOR2& Vector2)
```

Visual Basic

```
Public Declare Function Distance2D Lib "SetupDemo.dll"
(ByRef Vector1 As D3DVECTOR2, ByRef Vector2 As D3DVECTOR2)
As Single
```

Notice the ByRef and ByVal, you have to have them in there in order for it to work. Other than that it should work fine!

How To Use

You use the functions just as you would any other API.

Example:

```
Public Function Form_Load()
 Dim Distance As Single
 Dim Position1 As D3DVECTOR2
 Dim Position2 As D3DVECTOR2

 'Give some test values
 Position1.X = 50
 Position1.Y = 0
 Position2.X = 10
 Position2.Y = 0

 'Get the distance from the .dll
 Distance = Distance2D(Position1, Position2)
 'Display the distance
 Form1.Caption = Distance

 'The result should be 40
End Function
```

Please See The Demo's Included!!!

Conclusion

I hope this helped all the programmers who want to make a game engine and alike in Visual Basic. I tried my hardest to make it as easy as I could. If any information is incorrect please let me know. I learnt C++ and Visual Basic on my own, just started college, so some things may not be correct. Because of this, there is no warranty included with this. Use at your own risk! If your computer crashes due to you making .dll's from this tutorial, or your computer crashes from my tutorial/demo I am not held responsible. If you like this tutorial please vote for it and leave some feedback at http://www.planet-source-code.com . Thank you for the time in wanting to learn! You can never have to much knowledge!

If you need to contact me I can be reached at arielproductions@zoominternet.net

James Dougherty