Creating a Visual Basic .NET Module From a DLL Header File

- Create a Visual Basic module

- Declare library functions in this module using public declare … lib for each exported function

Example:

C (.h) File:

\_\_declspec(dllexport) int Add(int a, int b);

VB.NET (.vb) File:

Module MathDll

Public Declare Function Add Lib “Math.dll” (ByVal a As Integer, ByVal b As Integer) As Integer

End Module

- Create public const variables for each #define constant

Example:

C(.h) File:

#define PI 3.14159

#define START\_OF\_FRAME 0x55

VB.NET (.vb) File:

Module SomeDll

Public Const PI As Double = 3.14159

Public Const START\_OF\_FRAME As Byte = &H55

End Module

- Use the appropriate data types:

|  |  |  |
| --- | --- | --- |
| **Type** | **C** | **VB.NET** |
| 1 byte unsigned | bool, unsigned char, BYTE | Byte |
| 1 byte signed | char | SByte |
| 2 bytes unsigned | unsigned short, WORD | UShort |
| 2 bytes signed | short | Short |
| 4 bytes unsigned | unsigned int, unsigned long, UINT, DWORD | UInteger |
| 4 bytes signed | int, long, BOOL | Integer |
| 4 bytes floating point | float | Single |
| 8 bytes floating point | double | Double |
| 4/8 byte pointer | void\* | IntPtr |

See <http://msdn.microsoft.com/en-us/library/4xwz0t37(VS.80).aspx> for more information on data types.

- Special cases:

1. Visual Basic .NET differentiates between functions that return a value and functions that return void. A function that returns a value is called a function in VB, and a function that returns void (no return value) is called a subroutine.

Example:

**C (.h) File:**

\_\_declspec(dllexport) void SetLatch(int value);

**VB.NET (.vb) File:**

Module SomeDll

Public Declare Sub SetLatch Lib "Some.dll" (ByVal value As Integer)

End Module

1. Parameters passed as a pointer should use the ByRef keyword.

Example:

**C (.h) File:**

\_\_declspec(dllexport) void Halve(BYTE\* value);

**VB.NET (.vb) File:**

Module MathDll

Public Declare Sub Halve Lib "Math.dll" (ByRef value As Byte)

End Module

1. Parameters passed as a C string (char\* or LPSTR) should use the StringBuilder class.

Example:

**C (.h) File:**

\_\_declspec(dllexport) void GetName(char\* name, int size);

**VB.NET (.vb) File:**

Imports System.Text

Module SomeDll

Public Declare Sub GetName Lib "Some.dll" (ByVal name As StringBuilder, ByVal size As Integer)

End Module

Calling Example:

Dim name As New StringBuilder(100)

GetName(name, 100)

1. Parameters passed as an input C string (const char\* or LPCSTR) should use the String class.

Example:

**C (.h) File:**

\_\_declspec(dllexport) void SetName(const char\* name);

**VB.NET (.vb) File:**

Module SomeDll

Public Declare Sub SetName Lib “Some.dll” (name As String)

End Module

Calling Example:

Dim name As String = “John Smith”

SetName(name)

1. Parameters passed as an array should use a VB array of the appropriate type.

Example:

**C (.h) File:**

\_\_declspec(dllexport) void GetBuffer(BYTE\* buffer, int size, int\* bytesReturned);

**VB.NET (.vb) File:**

Module SomeDll

Public Declare Sub GetBuffer Lib "Some.dll" (ByVal buffer() As Byte, ByVal size As Integer, ByRef bytesReturned As Integer)

End Module

1. Parameters passed as a void pointer (void\*) should use the IntPtr type.

Example:

**C (.h) File:**

\_\_declspec(dllexport) void SetObject(void\* object);

\_\_declspec(dllexport) void GetObject(void\*\* object);

**VB.NET (.vb) File:**

Module SomeDll

Public Declare Sub SetObject Lib "Some.dll" (ByVal object As IntPtr)

Public Declare Sub GetObject Lib "Some.dll" (ByRef object As IntPtr)

End Module

Note: Passing pointers can be problematic when dealing with 32-bit/64-bit systems. IntPtr is platform dependent, meaning that it is a four byte pointer on 32-bit systems and an eight byte pointer on a 64-bit system. A .NET application running in 64-bit mode will not be able to load a 32-bit DLL. You must either build a separate 64-bit DLL or modify your .NET project to only run in 32-bit mode.

1. Structures must always be passed by reference in the C DLL.

Example:

**C (.h) File:**

typedef struct PERSON

{

BYTE id;

WORD month;

char name[10];

} PERSON, \*PPERSON;

\_\_declspec(dllexport) void GetPerson(PPERSON person);

\_\_declspec(dllexport) void SetPerson(PPERSON person);

**VB.NET (.vb) File:**

Imports System.Runtime.InteropServices

Module SomeDLL

<StructLayout(LayoutKind.Sequential)> Public Class PERSON

Public id As Byte

Public month As UShort

<MarshalAs(UnmanagedType.ByValArray, SizeConst := 10)> \_

Public name() As Byte

End Class

Public Declare Sub GetPerson Lib “Some.dll” (<[In](), Out()> ByVal person As PERSON)

Public Declare Sub SetPerson Lib “Some.dll” (ByVal person As PERSON)

End Module

Calling Example:

Dim person As New PERSON()

GetPerson(person)

person.id = 2

SetPerson(person)

Note: The “\_” character is a line continuation character in VB.NET, much like the “\” character in C++. The “[In]” attribute must be surrounded by square brackets, “[]”, and forces marshaling data from the caller to the callee. The “Out” attribute forces marshaling data from the callee back to the caller. The default attribute is “[In]”. If “Out” is specified, then “[In]” does not implicitly apply.

1. Parameters passed as an array of structs should use VB.NET arrays of **Structures** passed by value with the In/Out attributes specified as needed.

Examples:

**C (.h) File:**

typedef struct PERSON

{

BYTE id;

WORD month;

char name[10];

} PERSON, \*PPERSON;

\_\_declspec(dllexport) void GetPeople(PERSON people[], DWORD\* numPeople);

\_\_declspec(dllexport) void SetPeople(PERSON people[], DWORD numPeople);

**VB.NET (.vb) File:**

Imports System.Runtime.InteropServices

Module SomeDLL

<StructLayout(LayoutKind.Sequential)> Public Structure PERSON

Public id As Byte

Public month As UShort

<MarshalAs(UnmanagedType.ByValArray, SizeConst := 10)> \_

Public name() As Byte

End Structure

Public Declare Sub GetPeople Lib “Some.dll” (<[In](), Out()> ByVal people() As PERSON, ByRef numPeople As UInteger)

Public Declare Sub SetPeople Lib “Some.dll” (ByVal people() As PERSON, ByVal numPeople As UInteger)

End Module

Calling Example:

Dim people() As PERSON = New PERSON(0 To 1) {}

Dim numPeople As UInteger = people.Length

people(0).id = 1

people(0).month = 11

people(0).name = New Byte(0 To 9) {}

people(0).name(0) = &H31

people(1).id = 2

people(1).month = 12

people(1).name = New Byte(0 To 9) {}

people(1).name(0) = &H32

SetPeople(people, people.Length)

numPeople = people.Length

GetPeople(people, numPeople)

1. Callbacks must be defined as VB .NET delegates.

Examples:

**C (.h) File:**

typedef void (CALLBACK \*ProgressCallback)(int percent);

\_\_declspec(dllexport) void RegisterProgress(ProgressCallback progress);

**VB.NET (.vb) File:**

Module SomeDLL

Public Delegate Sub ProgressCallback(percent As Integer)

Public Declare Sub RegisterProgress Lib “Some.dll” (progress As ProgressCallback)

End Module

Calling Example:

Public Sub Progress(percent As Integer)

‘ do something with the progress percent

End Sub

Public Sub CallingExample()

RegisterProgress(AddressOf Progress)

End Sub

1. Win32 BOOL (4-byte int) data types can be automatically marshaled as a VB Boolean data type (1-byte).

Examples:

**C (.h) File:**

\_\_declspec(dllexport) void SetEnable(BOOL enable);

\_\_declspec(dllexport) void GetEnable(BOOL\* enable);

**VB.NET (.vb) File:**

Module SomeDLL

Public Declare Sub SetEnable Lib “Some.dll” (enable As Boolean)

Public Declare Sub GetEnable Lib “Some.dll” (ByRef enable As Boolean)

End Module

Calling Example:

Dim success As Boolean

Dim enable As Boolean = True

SetEnable(enable)

GetEnable(enable)

If enable Then

success = True

End If