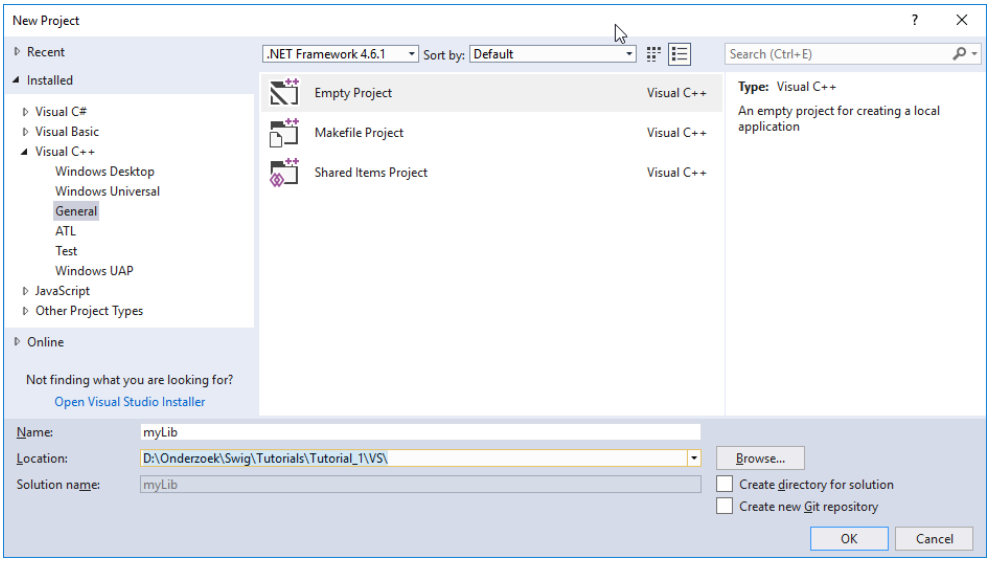
Contents

1 Tutorial 1 - C++ to C# using Visual Studio 2017 and SWIG

1.1 Create a C++ class

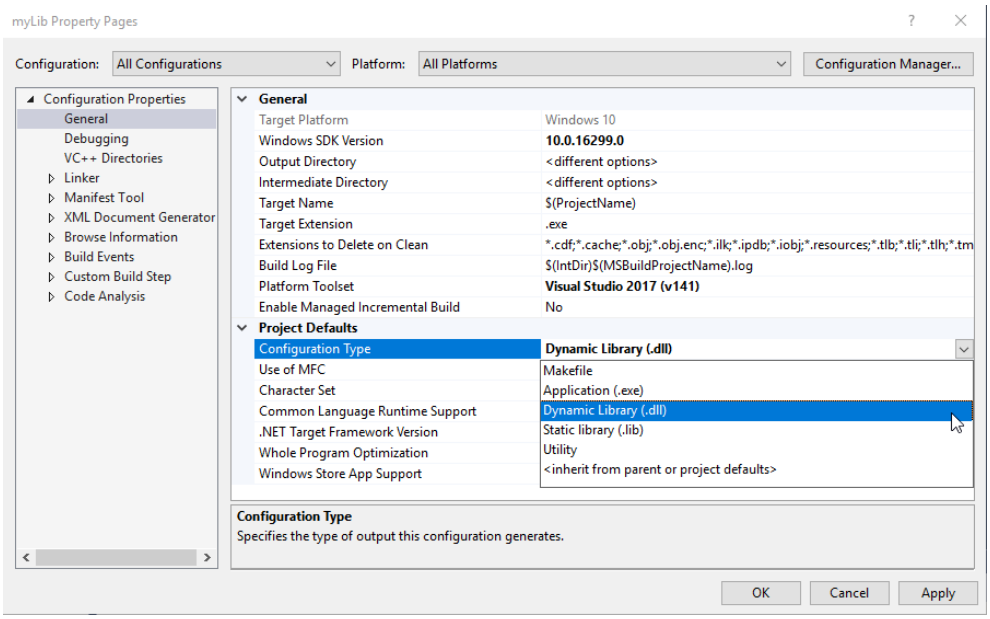
1.1.1 Start new Project

- Open Visual Studio
- Start a new *Visual C++* project
- *Empty Project*



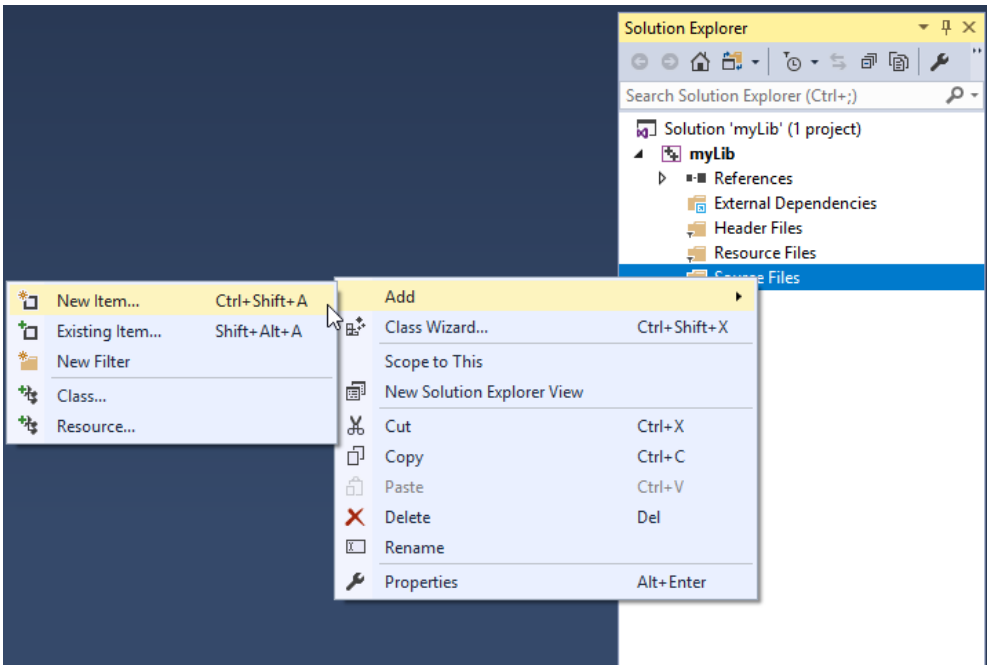
1.1.2 Set output to .dll

- Project → Properties

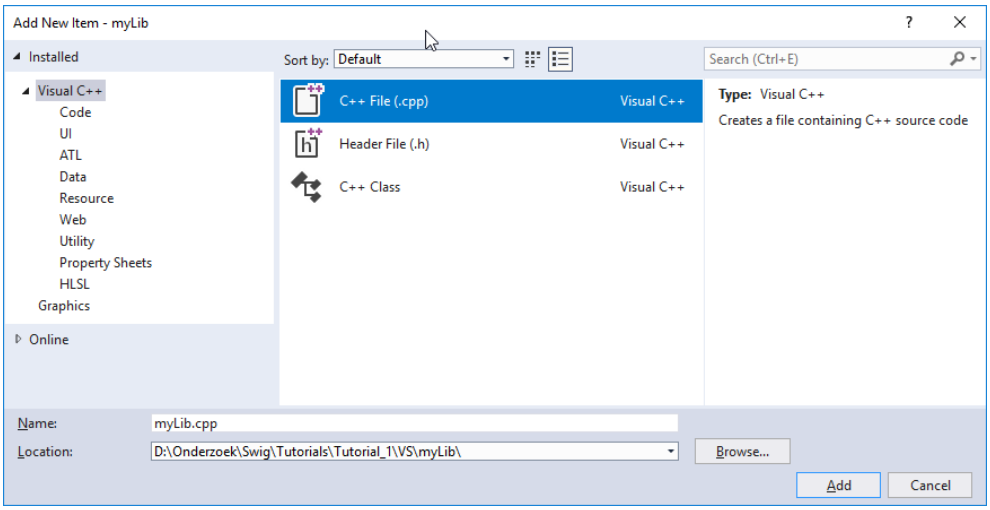


1.1.3 Create source file .cpp

- Add → New Item …



- myLib.cpp

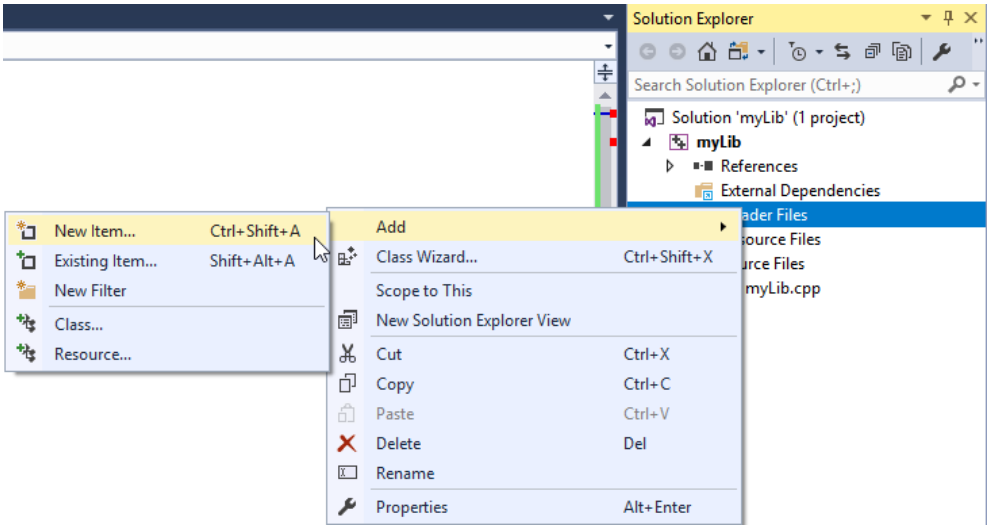


- Write the code of the library

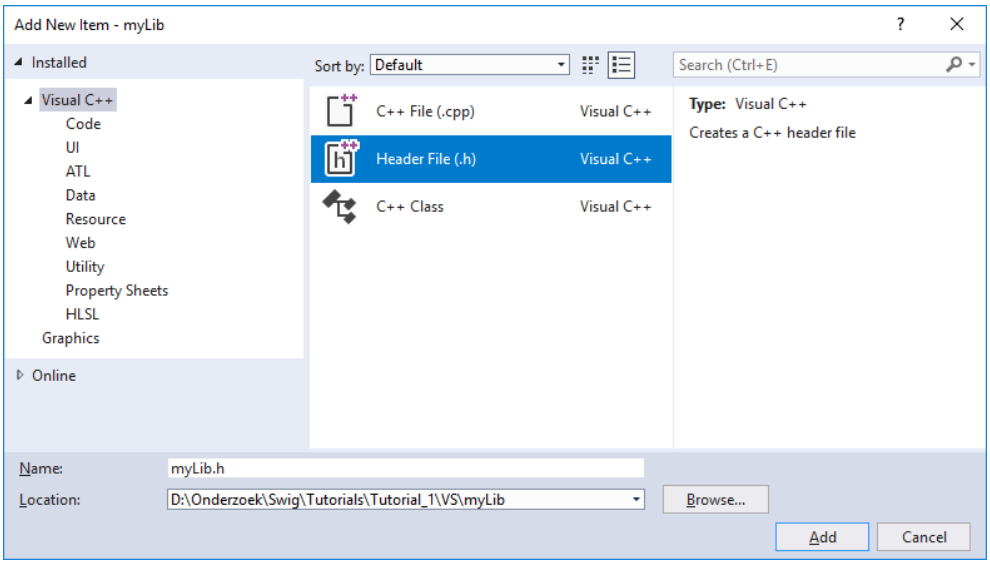
```
#include "myLib.h"

myLib::myLib(void)
{
        // Object being created
}

myLib::~myLib(void)
{
        // Object being deleted
}

int myLib::times2(int arg)
{
        return arg * 2;
}
```

1.1.4 Create header file .h

- Add → New Item …



- myLib.h



- Write the header file

```
#pragma once

class myLib
{
public:
        myLib(void);
        ~myLib(void);

        int times2(int arg);
};
```

- Make sure the class exports properly

> Use `__declspec(dllexport)` to export a function or method. You can export an entire C++ class by placing the `__declspec(dllexport)` before the class name, or you can export a single method by placing `__declspec(dllexport)` before the method name.

- Update the header file:

```
#pragma once

#define CLASS_DECLSPEC    __declspec(dllexport)

class CLASS_DECLSPEC myLib
{
public:
        myLib(void);
        ~myLib(void);

        int times2(int arg);
};
```

This header file is equal to:

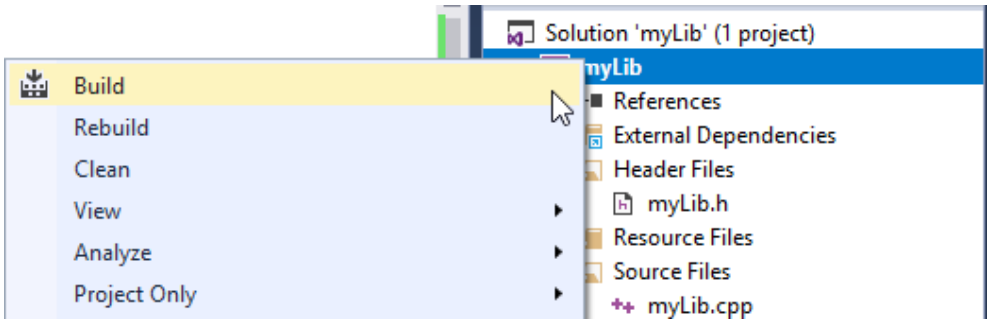
```
#pragma once

#define DLLExport    __declspec(dllexport)

class myLib
{
public:
        DLLExport myLib(void);
        DLLExport ~myLib(void);

        DLLExport int times2(int arg);
};
```

Other people suggest to use following code, but Microsoft discourages you to use this approach by giving a warning.

```
Warning C4273: 'myLib::myLib': inconsistent dll linkage
```

```
#pragma once

#ifdef _EXPORTING
#define CLASS_DECLSPEC    __declspec(dllexport)
#else
#define CLASS_DECLSPEC    __declspec(dllimport)
#endif

class CLASS_DECLSPEC myLib
{
public:
        myLib(void);
        ~myLib(void);

        int times2(int arg);
};
```

1.1.5 Check the C++ library by building it



Output:

```
1>------ Clean started: Project: myLib, Configuration: Debug Win32 ------
========== Clean: 1 succeeded, 0 failed, 0 skipped ==========
```

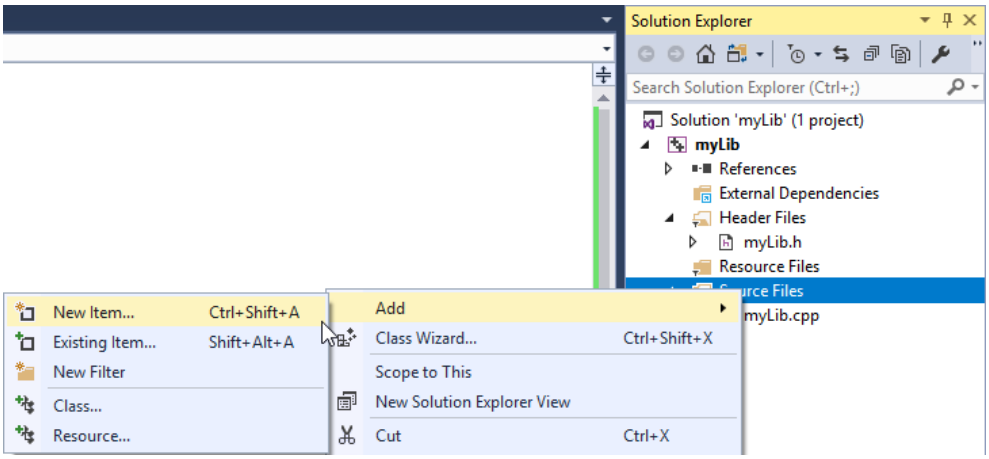
Since the build succeeded we can go on with our first tutorial.

1.2 Create a C# compatible class

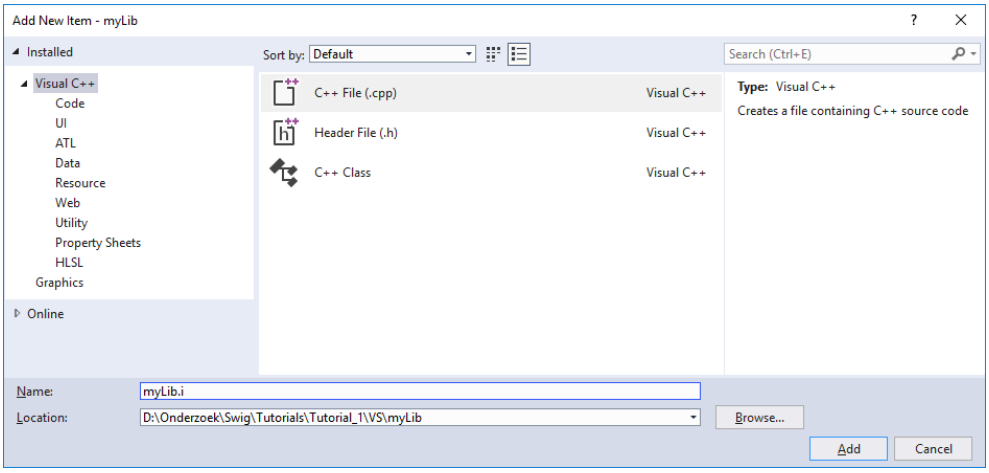
1.2.1 Create an interface file .i

In order to add the created source and header files to your favorite language (C#), you need to write an "interface file" which is the input to SWIG.

- Add → New Item …



- myLib.i

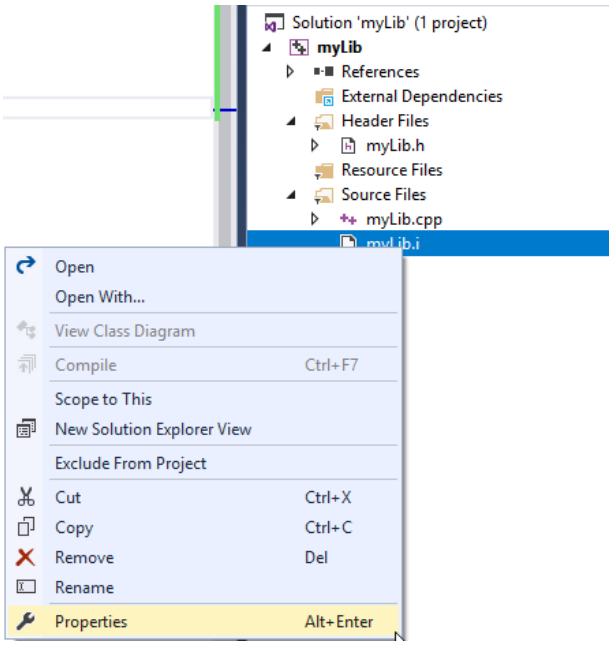


- Write the interface file

```
%module tutorial

%{
        #include "myLib.h"
%}

%include <windows.i>
%include "myLib.h"
```

1.2.2 Set a custom build tool for SWIG

- myLib.i → Properties



- Configuration Properties → General →Item Type →Custom Build Tool
- Apply

Now you get a new item on the left side *Custom Build Tool*

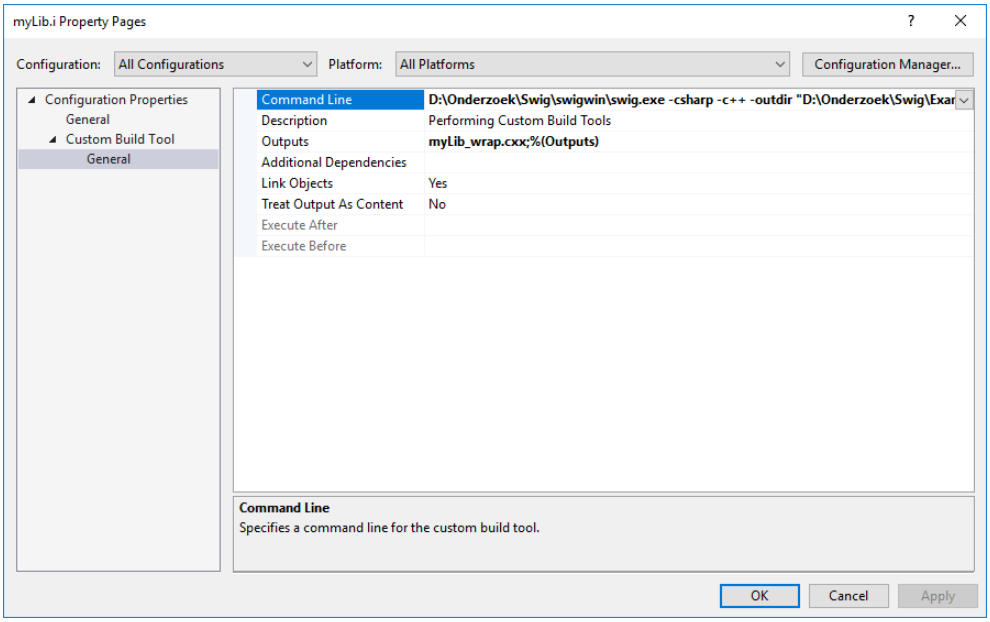
- Configuration Properties → Custom Build Tool → General → Command Line → …
- Type following line of code:

```
C:\dev\swigwin\swig.exe -csharp -c++ -outdir
"D:\Onderzoek\Swig\Tutorials\Tutorial_1\VS\myLib\Generated" myLib.i
```

- Configuration Properties → Custom Build Tool → General → Outputs → …

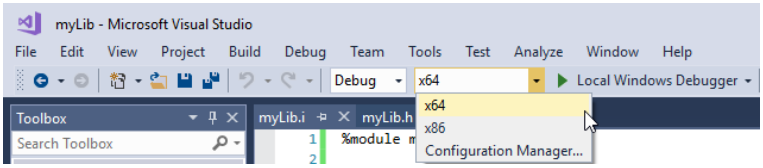
```
myLib_wrap.cxx;%(Outputs)
```

- Apply



1.2.3 Compile the interface file

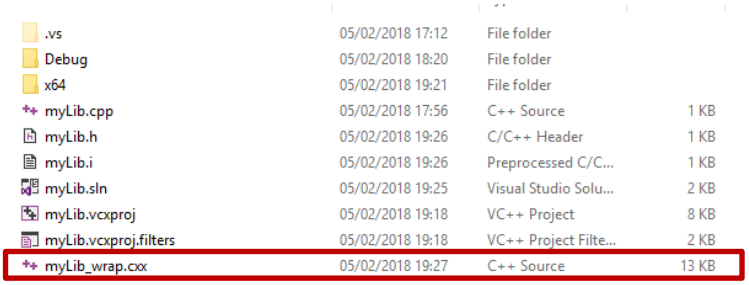
- Change the building platform to x64



- myLib.i → Compile (this compiles the library using the swig compiler)

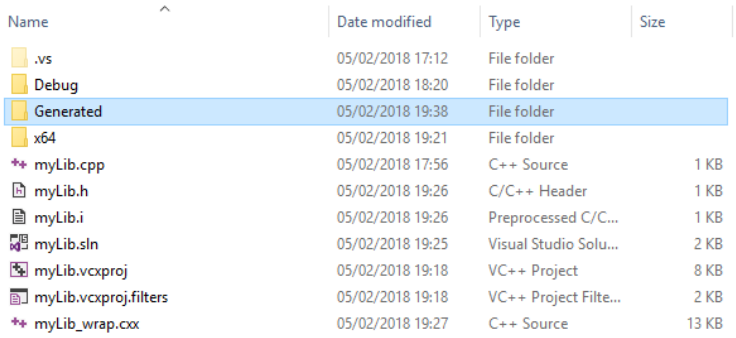
```
1>------ Build started: Project: myLib, Configuration: Debug x64 ------
1>Performing Custom Build Tools
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

If you now look in the project folder you will see a new file 'myLib_wrap.cxx', created by the SWIG compiler.

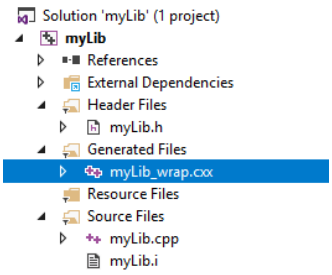


1.2.4 Build the C# library

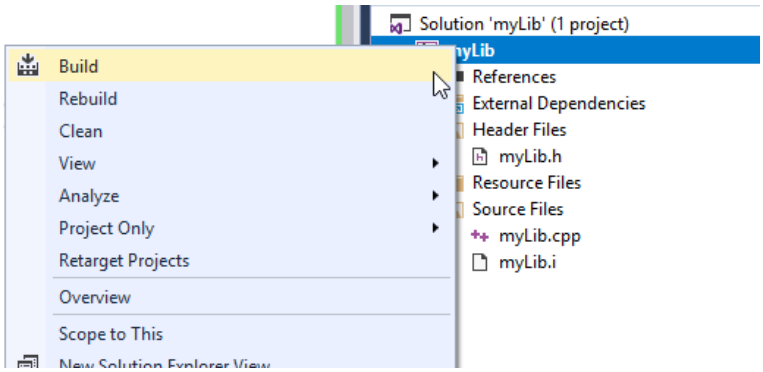
- It is now time to build the C# library
- First we create the folder where the C# files will be saved. We defined this path earlier in the custom build tool 'D:\Onderzoek\Swig\Tutorials\Tutorial_1\VS\myLib\Generated'.



- It is important to add the compiled .cxx to the solution in order to create a correct .dll library.

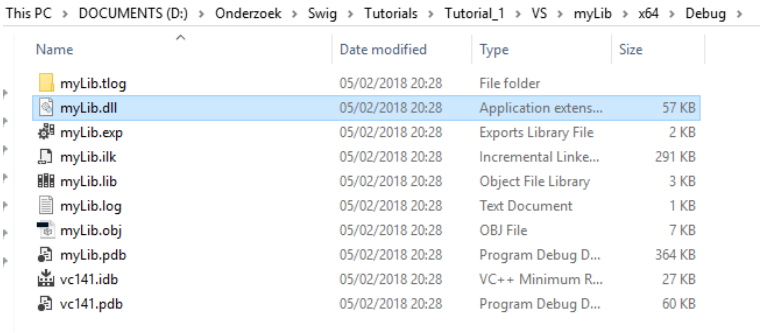


- Now we build the entire cpp project such that we have a C# library.

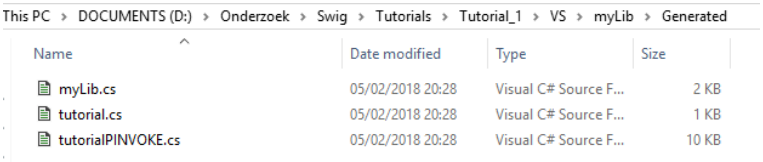



```
1>------ Rebuild All started: Project: myLib, Configuration: Debug x64 ------
1>Performing Custom Build Tools
1>myLib_wrap.cxx
1>myLib.cpp
1>Generating Code...
1>   Creating library D:\Onderzoek\Swig\Tutorials\tmp\myLib\x64\Debug\myLib.lib
and object D:\Onderzoek\Swig\Tutorials\tmp\myLib\x64\Debug\myLib.exp
1>myLib.vcxproj -> D:\Onderzoek\Swig\Tutorials\tmp\myLib\x64\Debug\myLib.dll
========== Rebuild All: 1 succeeded, 0 failed, 0 skipped ==========
```

Congratulations! We now have a .dll and .lib in the 'Debug' folder

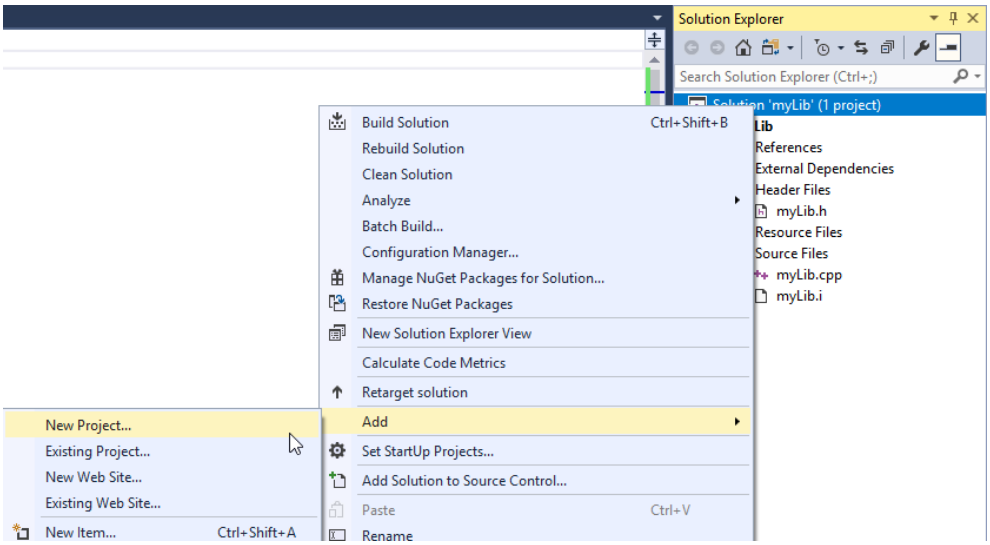


And we have a myLib.cs, tutorial.cs and tutorialPINVOKE.cs in the 'Generated' folder

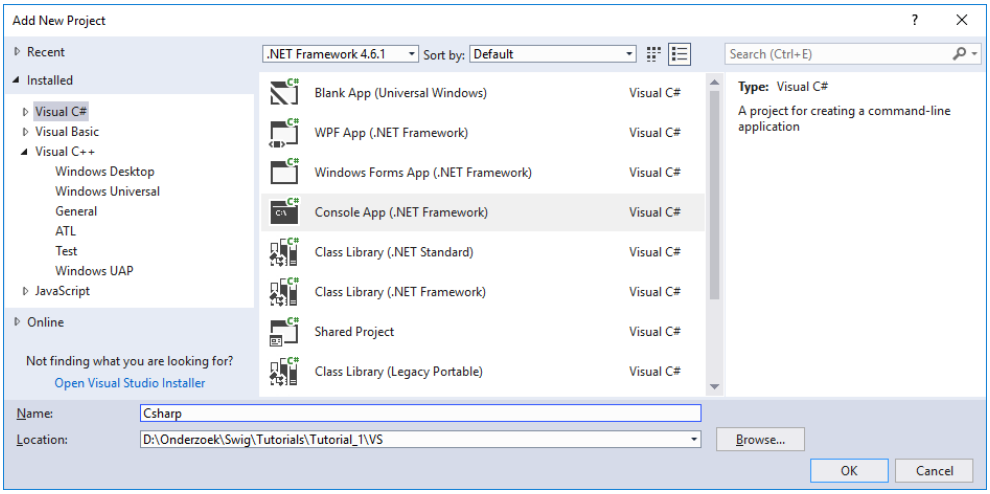


1.3 Build a C# test console application

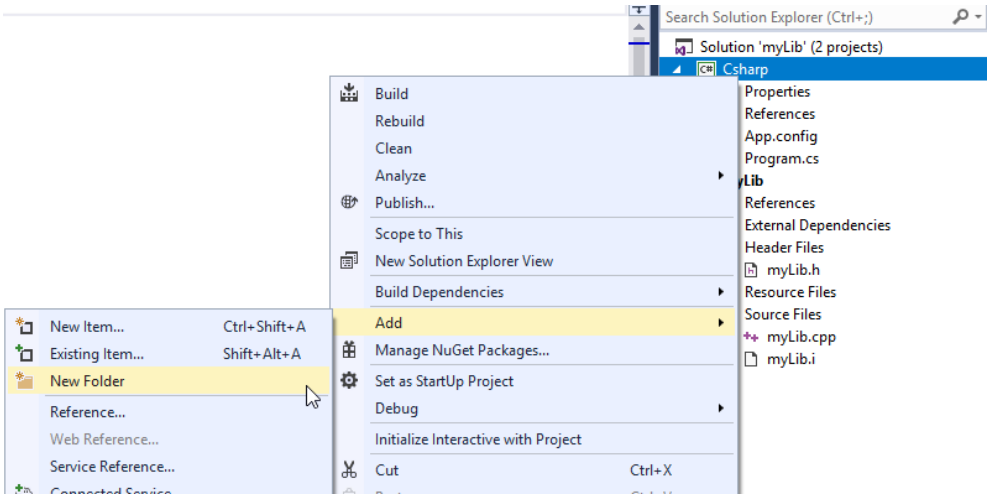
- Add → New project …



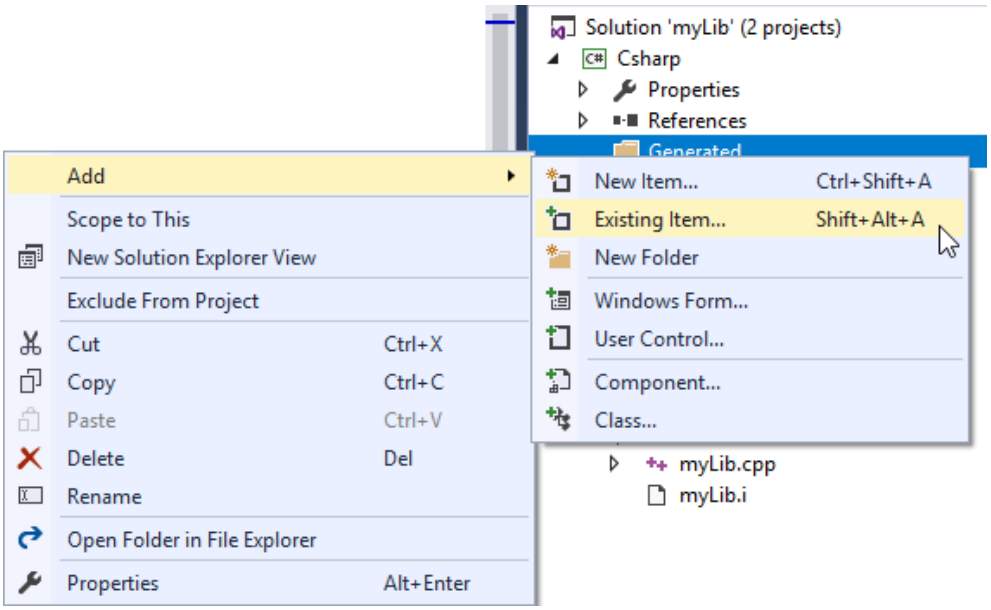
- Visual C# → Console App (.NET Framework)



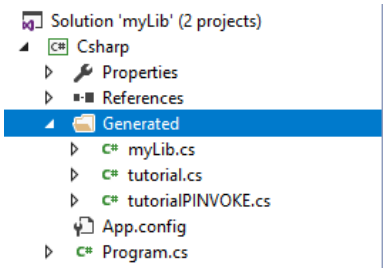
- Now add a new folder to the project where we you can put the .cs library files that we are created in paragraph 1.2.4.



- Add the .cs files



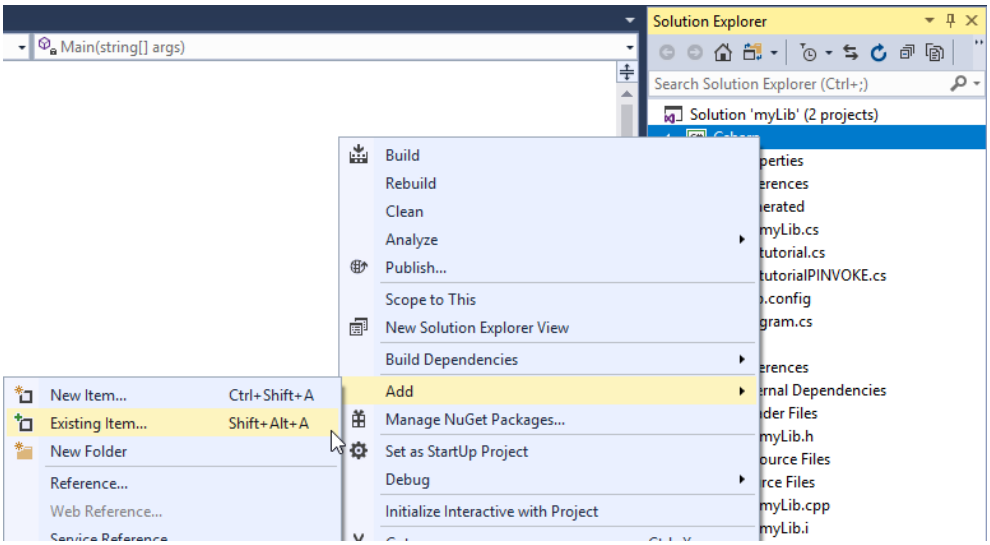
Now the project looks like this:



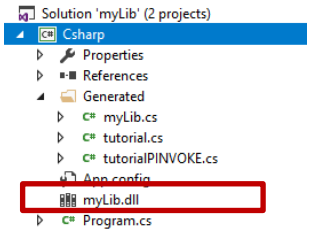
- Write the main program

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Csharp
{
    class Program
    {
        static void Main(string[] args)
        {
            myLib myMathLib = new myLib();
            Console.WriteLine(myMathLib.times2(48));
            Console.ReadLine();
        }
    }
}
```

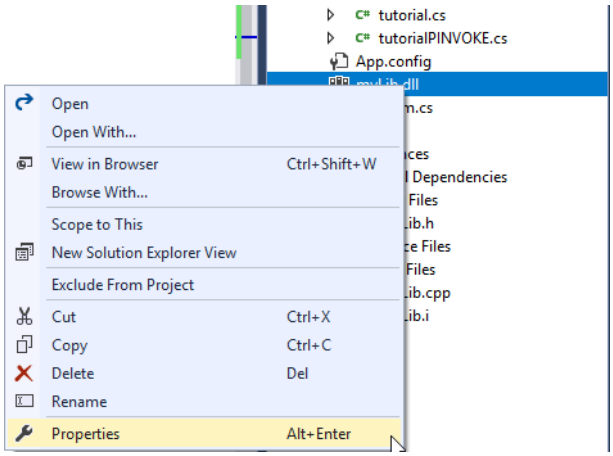
- Add the dll-file created in paragraph 1.2.4



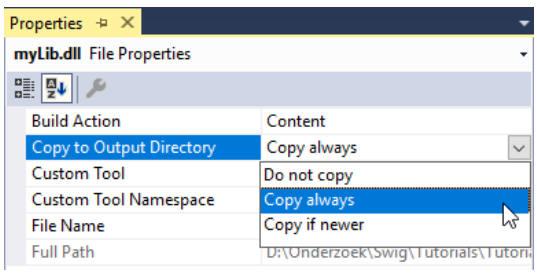
You can now see the .dll file in the project structure



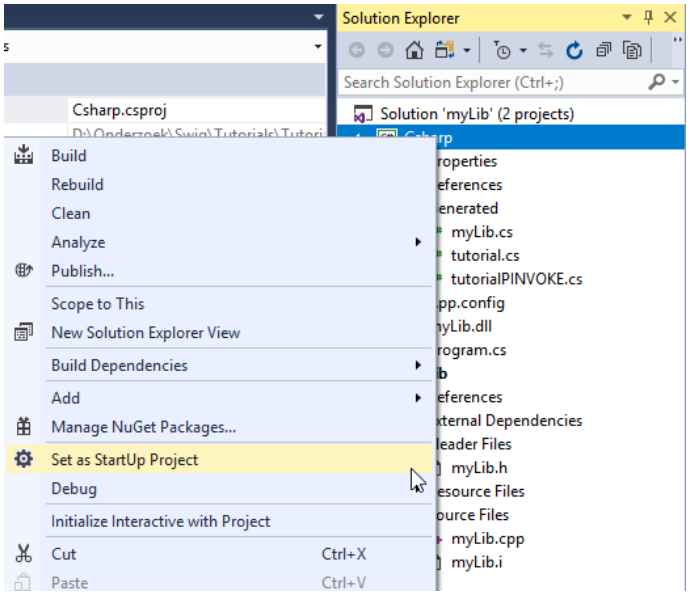
- myLib.dll → Properties



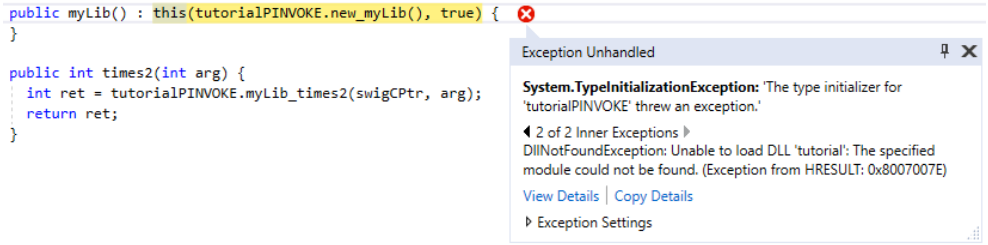
- Copy to Output Directory → Copy Always



- CSharp → Set as Startup Project



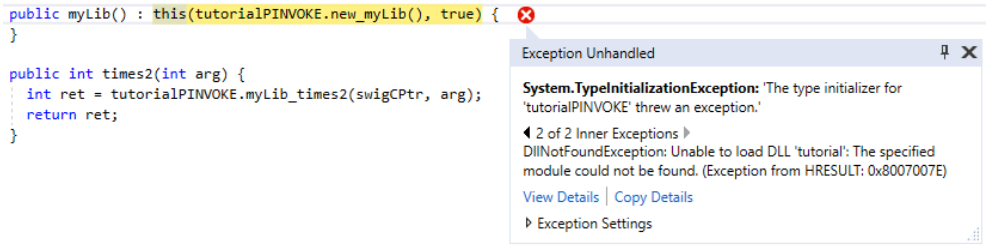
- If you compile now you will get the following error:



The reason is that the C# program expects the dll-file to be named ‘tutorial.dll’. And currently it is named ‘myLib.dll’.

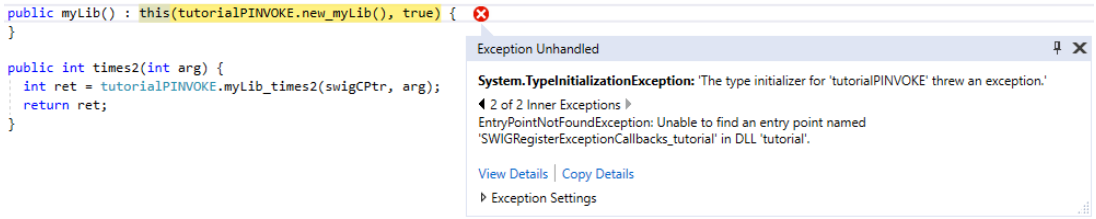
1.4 Possible errors

1.4.1 Dll naming error



The reason is that the C# program expects the dll-file to be named ‘tutorial.dll’. And currently it is named ‘myLib.dll’.

1.4.2 EntryPointNotFoundException



This error looks a bit confusion but it just means that you forgot to add the .cxx file in your C++ project. The .dll file is compiled without having the correct content.

So it is important that the C++ solution looks like this: