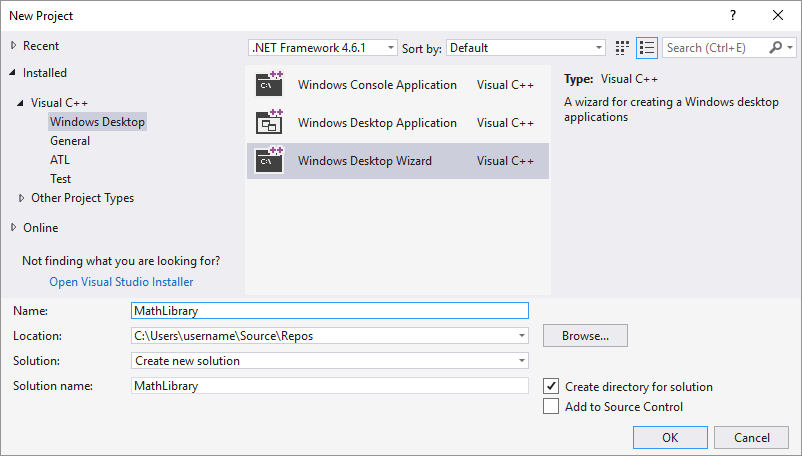
**To create a DLL project in Visual Studio 2017 version 15.3 or later**

1. On the menu bar, choose **File** > **New** > **Project** to open the **New Project** dialog box.
2. In the left pane of the **New Project** dialog box, expand **Installed** and **Visual C++** if necessary, and then choose **Windows Desktop**. In the center pane, select **Windows Desktop Wizard**. Enter MathLibrary in the **Name** box to specify a name for the project.



1. Choose the **OK** button to dismiss the **New Project** dialog and start the **Windows Desktop Project** wizard.
2. In the **Windows Desktop Project** wizard, under **Application type**, select **Dynamic Link Library (.dll)**.
3. Choose the **OK** button to create the project.

Note

Additional steps are required to fix an issue in Visual Studio 2017 version 15.3. Follow these instructions to see if you need to make this change.

1. In **Solution Explorer**, if it is not already selected, select the **MathLibrary** project under **Solution 'MathLibrary'**.
2. On the menu bar, choose **Project** > **Properties**.
3. In the left pane of the **Property Pages** dialog box, select **Preprocessor** under **Configuration Properties** > **C/C++**. Check the contents of the **Preprocessor Definitions** property.  
   If you see **MATHLIBRARY\_EXPORTS** in the **Preprocessor Definitions** list, then you do not need to change anything. If you see **MathLibrary\_EXPORTS** instead, then continue to follow these steps.
4. At the top of the **Property Pages** dialog, change the **Configuration** drop-down to **All Configurations**.
5. In the property pane, select the drop-down control next to the edit box for **Preprocessor Definitions**, and then choose **Edit**.
6. In the top pane of the **Preprocessor Definitions** dialog, add a new symbol, MATHLIBRARY\_EXPORTS.
7. Choose **OK** to dismiss the **Preprocessor Definitions** dialog, and then choose **OK** to save your changes to the project properties.

**To create a DLL project in older versions of Visual Studio**

1. On the menu bar, choose **File** > **New** > **Project**.
2. In the left pane of the **New Project** dialog box, expand **Installed** > **Templates**, and select **Visual C++**, and then in the center pane, select **Win32 Console**[**Application**](https://docs.microsoft.com/en-us/cpp/build/walkthrough-creating-and-using-a-dynamic-link-library-cpp?view=vs-2019). Enter MathLibrary in the **Name** edit box to specify a name for the project.
3. Choose the **OK** button to dismiss the **New Project** dialog and start the **Win32 Application Wizard**.
4. Choose the **Next** button. On the **Application Settings** page, under **Application type**, select **DLL**.
5. Choose the **Finish** button to create the project.

When the wizard completes the solution, you can see the generated project and source files in the **Solution Explorer** window in Visual Studio.

Right now, this DLL doesn't do very much. Next, you create a header file to declare the functions your DLL exports, and then add the function definitions to the DLL to make it more useful.

**To add a header file to the DLL**

1. To create a header file for your functions, on the menu bar, choose **Project** > **Add New Item**.
2. In the **Add New Item** dialog box, in the left pane, select **Visual C++**. In the center pane, select **Header File (.h)**. Specify MathLibrary.h as the name for the header file.
3. Choose the **Add** button to generate a blank header file, which is displayed in a new editor window.
4. Replace the contents of the header file with this code:

// MathLibrary.h - Contains declarations of math functions

#pragma once

#ifdef MATHLIBRARY\_EXPORTS

#define MATHLIBRARY\_API \_\_declspec(dllexport)

#else

#define MATHLIBRARY\_API \_\_declspec(dllimport)

#endif

extern "C" MATHLIBRARY\_API double Add(double a, double b);

extern "C" MATHLIBRARY\_API double Multiply(double a, double b);

extern "C" MATHLIBRARY\_API double AddMultiply(double a, double b);

Notice the preprocessor statements at the top of the file. By default, the New Project template for a DLL adds ***PROJECTNAME*\_EXPORTS** to the defined preprocessor macros for the DLL project. In this example, Visual Studio defines **MATHLIBRARY\_EXPORTS** when your MathLibrary DLL project is built. (The wizard in Visual Studio 2017 version 15.3 doesn't force this symbol definition to upper case. If you name your project "MathLibrary", then the symbol defined is MathLibrary\_EXPORTS instead of MATHLIBRARY\_EXPORTS. That's why there are extra steps above to add this symbol.)

When the **MATHLIBRARY\_EXPORTS** macro is defined, the **MATHLIBRARY\_API** macro sets the \_\_declspec(dllexport) modifier on the function declarations. This modifier tells the compiler and linker to export a function or variable from the DLL so that it can be used by other applications. When **MATHLIBRARY\_EXPORTS**is undefined, for example, when the header file is included by a client application, **MATHLIBRARY\_API** applies the \_\_declspec(dllimport) modifier to the declarations. This modifier optimizes the import of the function or variable in an application. For more information, see [dllexport, dllimport](https://docs.microsoft.com/en-us/cpp/cpp/dllexport-dllimport?view=vs-2019).

**To add an implementation to the DLL**

1. In the editor window, select the tab for **MathLibrary.cpp** if it's already open. If not, in **Solution Explorer**, open **MathLibrary.cpp** in the **Source Files** folder of the **MathLibrary** project.
2. In the editor, replace the contents of the MathLibrary.cpp file with the following code:

C++Copy

// MathLibrary.cpp : Defines the exported functions for the DLL.

#include "pch.h"

#include "framework.h"

#include "MathLibrary.h"

MATHLIBRARY\_API double Add(double a, double b) { return a + b; }

MATHLIBRARY\_API double Multiply(double a, double b) { return a \* b; }

MATHLIBRARY\_API double AddMultiply(double a, double b) { return a + (a \* b); }

To verify that everything works so far, compile the dynamic link library. To compile, choose **Build** > **Build Solution** on the menu bar. The output should look something like:

OutputCopy

1>------ Build started: Project: MathLibrary, Configuration: Debug Win32 ------

1>stdafx.cpp

1>MathLibrary.cpp

1>dllmain.cpp

1>Generating Code...

1> Creating library C:\Users\username\Source\Repos\MathLibrary\Debug\MathLibrary.lib and object C:\Users\username\Source\Repos\MathLibrary\Debug\MathLibrary.exp

1>MathLibrary.vcxproj -> C:\Users\username\Source\Repos\MathLibrary\Debug\MathLibrary.dll

1>MathLibrary.vcxproj -> C:\Users\username\Source\Repos\MathLibrary\Debug\MathLibrary.pdb (Partial PDB)

========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========

Congratulations, you've created a DLL using Visual C++! Next, you'll create a client app that uses the functions exported by the DLL.

**Create a client app that uses the DLL**

When you create a DLL, you must think about how your DLL can be used. To compile code that calls the functions exported by a DLL, the declarations must be included in the client source code. At link time, when these calls to DLL functions are resolved, the linker must have an *import library*, a special library file that contains information for Windows about how to find the functions, instead of the actual code. And at run time, the DLL must be available to the client, in a location that the operating system can find.

To make use of a DLL, whether your own or a third-party DLL, your client app project must find the headers that declare the DLL exports, the import libraries for the linker, and the DLL itself. One way, is to copy all of these files into your client project. For third-party DLLs that are unlikely to change while your client is in development, this method may be the best way to use them. However, when you also build the DLL, it's better to avoid duplication. If you make a copy of DLL files that are under development, you may accidentally change a header file in one copy but not the other, or use an out-of-date library. To avoid this problem, we recommend you set the include path in your client project to include the DLL header files from the DLL project. Also, set the library path in your client project to include the DLL import libraries from the DLL project. And finally, copy the built DLL from the DLL project into your build output directory. This step allows your client app to use the same DLL code you build.

**To create a client app in Visual Studio 2017 version 15.3 or later**

1. To create a C++ app that uses the DLL that you created, on the menu bar, choose **File** > **New** > **Project**.
2. In the left pane of the **New Project** dialog, select **Windows Desktop** under **Installed** > **Visual C++**. In the center pane, select **Windows Desktop Wizard**. Specify the name for the project, MathTest, in the **Name**edit box.
3. Choose **OK** to start the **Windows Desktop Project** wizard. In the wizard, choose **OK** to create the client app project.

**To create a client app in older versions of Visual Studio 2017**

1. To create a C++ app that uses the DLL that you created, on the menu bar, choose **File** > **New** > **Project**.
2. In the left pane of the **New Project** dialog, select **Win32** under **Installed** > **Templates** > **Visual C++**. In the center pane, select **Win32 Console Application**. Specify the name for the project, MathTest, in the **Name**edit box.
3. Choose the **OK** button to dismiss the **New Project** dialog and start the **Win32 Application Wizard**. On the **Overview** page of the **Win32 Application Wizard** dialog box, choose the **Next** button.
4. On the **Application Settings** page, under **Application type**, select **Console application** if it isn't already selected.
5. Choose the **Finish** button to create the project.

When the wizard finishes, a minimal console application project is created for you. The name for the main source file is the same as the project name that you entered earlier. In this example, it's named **MathTest.cpp**. You can build it, but it doesn't use your DLL yet.

Next, to call the MathLibrary functions in your source code, your project must include the MathLibrary.h file. You could copy this header file into your client app project, then add it to the project as an existing item. This method can be a good choice for third-party libraries. However, if you're working on the code for your DLL at the same time as your client, that might lead to changes in one header file that aren't shown in the other. To avoid this issue, you can change the **Additional Include Directories** path in your project to include the path to the original header.

**To add the DLL header to your include path**

1. Open the **Property Pages** dialog box for the **MathTest** project.
2. In the **Configuration** drop-down box, select **All Configurations** if it isn't already selected.
3. In the left pane, select **General** under **Configuration Properties** > **C/C++**.
4. In the property pane, select the drop-down control next to the **Additional Include Directories** edit box, and then choose **Edit**.
5. Double-click in the top pane of the **Additional Include Directories** dialog box to enable an edit control.
6. In the edit control, specify the path to the location of the **MathLibrary.h** header file. In this case, you can use a relative path from the folder that contains your .cpp files in the client project to the folder that contains the .h file in the DLL project. If your client project is in a separate solution in the same folder as the DLL solution, the relative path should look like this:

..\..\MathLibrary\MathLibrary

If your DLL and client projects are in the same solution, or the solutions are in different folders, then you must adjust the relative path accordingly.

1. Once you've entered the path to the header file in the **Additional Include Directories** dialog box, choose the **OK** button to go back to the **Property Pages** dialog box, and then choose the **OK** button to save your changes.

You can now include the **MathLibrary.h** file and use the functions it declares in your client application. Replace the contents of **MathTest.cpp** by using this code:

C++Copy

// MathTest.cpp : This file contains the 'main' function. Program execution begins

// and ends there.

\

#include <iostream>

#include <windows.h>

typedef double (\*funcAdd)(double, double);

typedef double (\*funcMult)(double, double);

int main()

{

HINSTANCE hDLL;

funcAdd Add;

funcMult Multiply;

const wchar\_t \*libName = L"MathLibrary";

hDLL = LoadLibraryEx(libName, NULL, NULL); // Handle to DLL

if (hDLL != NULL) {

Add = (funcAdd)GetProcAddress(hDLL, "Add");

Multiply = (funcMult)GetProcAddress(hDLL, "Multiply");

if (Add != NULL) {

std::cout << "10 + 10 = " << Add(10, 10) << std::endl;

}

else

std::cout << "Did not load Add correctly." << std::endl;

if (Multiply != NULL) {

std::cout << "50 \* 10 = " << Multiply(50, 10) << std::endl;

}

else

std::cout << "Did not load Multiply correctly." << std::endl;

FreeLibrary(hDLL);

}

else {

std::cout << "Library load failed!" << std::endl;

}

std::cin.get();

return 0;

}This code can be compiled, but not linked, because the linker can't find the import library required to build the app yet. The linker must find the MathLibrary.lib file to link successfully. Add the MathLibrary.lib file to the build by setting the **Additional Dependencies** property. Once again, you could copy the library file into your client app project, but if both the library and the client app are under development, that might lead to changes in one copy that aren't shown in the other. To avoid this issue, you can change the **Additional Library Directories** path in your project to include the path to the original library when you link.

**To add the DLL import library to your project**

1. Open the **Property Pages** dialog box for the **MathTest** project.
2. In the **Configuration** drop-down box, select **All Configurations** if it isn't already selected.
3. In the left pane, select **Input** under **Configuration Properties** > **Linker**. In the property pane, select the drop-down control next to the **Additional Dependencies** edit box, and then choose **Edit**.
4. In the **Additional Dependencies** dialog, add MathLibrary.lib to the list in the top edit control.
5. Choose **OK** to go back to the **Property Pages** dialog box.
6. In the left pane, select **General** under **Configuration Properties** > **Linker**. In the property pane, select the drop-down control next to the **Additional Library Directories** edit box, and then choose **Edit**.
7. Double-click in the top pane of the **Additional Library Directories** dialog box to enable an edit control. In the edit control, specify the path to the location of the **MathLibrary.lib** file. Enter this value to use a macro that works for both Debug and Release builds:

..\..\MathLibrary\$(IntDir)

1. Once you've entered the path to the library file in the **Additional Library Directories** dialog box, choose the **OK** button to go back to the **Property Pages** dialog box.

Your client app can now compile and link successfully, but it still doesn't have everything it needs to run. When the operating system loads your app, it looks for the MathLibrary DLL. If it can't find the DLL in certain system directories, the environment path, or the local app directory, the load fails. One way to avoid this issue is to copy the DLL to the directory that contains your client executable as part of the build process. To copy the DLL, you can add a **Post-Build Event** to your project, to add a command that copies the DLL to your build output directory. The command specified here copies the DLL only if it's missing or has changed, and uses macros to copy to and from the correct Debug or Retail locations for your configuration.

**To copy the DLL in a post-build event**

1. Open the **Property Pages** dialog box for the **MathTest** project if it isn't already open.
2. In the Configuration drop-down box, select **All Configurations** if it isn't already selected.
3. In the left pane, select **Post-Build Event** under **Configuration Properties** > **Build Events**.
4. In the property pane, select the edit control in the **Command Line** field, and then enter this command:

xcopy /y /d "..\..\MathLibrary\$(IntDir)MathLibrary.dll" "$(OutDir)"

1. Choose the **OK** button to save your changes to the project properties.

Now your client app has everything it needs to build and run. Build the application by choosing **Build** > **Build Solution** on the menu bar. The **Output** window in Visual Studio should have something like:

OutputCopy

1>------ Build started: Project: MathTest, Configuration: Debug Win32 ------

1>pch.cpp

1>MathTest.cpp

1>MathTest.vcxproj -> C:\Users\username\Source\Repos\MathTest\Debug\MathTest.exe

1>MathTest.vcxproj -> C:\Users\username\Source\Repos\MathTest\Debug\MathTest.pdb (Partial PDB)

1>1 File(s) copied

========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========

Congratulations, you've created an application that calls functions in your DLL. Now run your application to see what it does. On the menu bar, choose **Debug** > **Start Without Debugging**. Visual Studio opens a command window for the program to run in.

C:\Users\ndung\OneDrive\Desktop\C++ REVIEWS\MapDLL\x64\Debug