Creating a C Library in Visual Studio 2019

This document provides step-by-step instructions for creating a Windows static library in C using Visual Studio 2019. A static library is a .LIB file containing a collection one or more object files. When an application links to a library, the parts of the library it uses become a part of that application's executable file. Thus, when several applications link to a library, each gets its own copy of object code used from the library.

Throughout this document, the term *user* refers to another programmer making use of our library.

Creating the Library Project

Here are the steps required to create a library in Visual Studio 2019.

1. On the **File** menu, click **New | Project**.
2. In the **Create a New Project** window:
    a. In the **Language** list near the top of the window, select **C++** (even though we're working with C).
    b. In the **Platform** list, select **Windows**.
    c. In the **Project type** list, select **Library**.
3. Scroll through the resulting templates to find one labeled **Windows Desktop Wizard** (even if this library will not be associated with desktop applications). Click on that template, and click **Next**.
    a. If you don't see this template in the list, you may need to reconfigure your Visual Studio installation to include it.
4. In the **Configure your new project** window,
    a. In the **Project name** box, type the name of your new library project.
    b. If you want the project to be located somewhere other than the location shown in the **Location** box, type the desired location, or click the … ellipsis button to navigate to the desired location.
    c. If you want the solution name to be different from the project name, type the solution name in the **Solution name** box. For example, if you want to create one solution that potentially contains multiple libraries, you could name the project ***MyFirstLibrary*** and name the solution ***MyLibraryCollection***.
    d. Leave unchecked the **Place solution and project in the same directory** checkbox.

    e. When you're done configuring your project, click **Create**.
5. In the **Windows Desktop Project** window:
    a. Under **Application Type**, select **Static Library (.lib)**.
    b. Under **Additional Options**, make sure all the boxes are unchecked, and then click to check the **Empty project** box.
    c. Click **OK**.
6. In the **Solution Explorer** panel, you should see the library project (e.g., MyFirstLibrary) under the solution (e.g., MyLibraryCollection).
    a. If you don't see the **Solution Explorer** panel, then on the **View** menu, click **Solution Explorer**.
    b. You can confirm that your project is configured to generate a static library .LIB file using the following steps:
        i. Right-click the project name, and click **Properties**.
        ii. In the left pane of the **Property Pages** window, navigate to **Configuration Properties | General**.
        iii. In the right pane of the **Property Pages** window, under **General Properties**, you should see the **Configuration Type** set to **Static library (.lib)**.

At this point, you have a Visual Studio project that will build a library (.LIB file). The next step involves adding source code to the library.

Adding a Header File to the Library

The users of your library will need a header file to include, containing the prototypes of functions you're making available, as well as any preprocessor symbols, typedefs, structs, unions, etc. So, the first order of business after creating your library project is creating this library's header file. Although you can certainly have more than one header file in your library, there should be just one header file that is designated for inclusion by users of your library.

1. In **Solution Explorer**, find your library project, and under that project, right-click the **Header Files** folder.
2. On the context menu, click **Add | New Item…**.
3. In the left panel of the **Add New Item** window, click to select **Visual C++**.
4. In the center panel of the **Add New Item** window, click **Header File (.h)**.
5. In the **Name** box, type the name of your new C header file, ensuring that it has the **.H** file name extension.
6. Click **Add**. The new empty C header file will be opened in the code editor panel.
7. Although we're currently only interested in allowing the library functions to be called from C user code, it is best practice to allow C++ user code to call functions in our library. Calling our library functions from C++ (or from other languages) requires that our library function prototypes be declared as **extern "C"**. This avoids C++ "name mangling." As long as the user's code includes your supplied header file with the **extern "C"** declaration, the compiler of the user code will be able to generate the correct call to your library function. The easiest way to do this, especially if you have more than one visible function in your library, is to surround your collection of function prototypes with

an **extern "C"** block:

```
#ifdef __cplusplus
extern "C" {
#endif

    // All visible library function prototypes go here.

#ifdef __cplusplus
}
#endif
```

Notice that the extern declaration/block opening and the block closing are both surrounded by a preprocessor guard based on the definition of **__cplusplus**. This predefined C++ macro (which begins with two underscores) is defined when using a C++ compiler, and is not defined when using a C compiler.

Adding a C Source File to the Library

1. In **Solution Explorer**, find your library project, and under that project, right-click the **Source Files** folder.
2. On the context menu, click **Add | New Item....**
3. In the left panel of the **Add New Item** window, click to select **Visual C++**.
4. In the center panel of the **Add New Item** window, click **C++ File (.cpp)**.
5. In the **Name** box, type the name of your new C source code file, and be sure to change the default file extension from the **.CPP** extension to **.C** extension.
    - Using the proper source file extension is very important. The .C file extension will cause the compiler to use C syntax and semantics when compiling your code. If you use the .CPP file extension, the compiler will use C++ syntax and semantics.
    - Keep in mind that you're creating a library, so your library will *not* have its own main function. Applications that link to your library will supply the program's main function.
6. Click **Add**. The new empty C source code file will be opened in the code editor panel.
7. Because your source file has the .C extension, you don't need to surround your function definitions with **extern "C"**, because the C compiler will, by definition, not perform C++ name mangling. However, if your source file had a .CPP extension, then the C function definitions would need to be placed within an **extern "C"** block:

```
extern "C"
{
    // All visible C library function prototypes go here.
}
```

You can add as many source files as you wish to your library project. Be sure that any .C source files which define and/or make use of the visible functions whose prototypes appear in your .H file actually includes the .H file, so that you can use the compiler to ensure that the header file and library implementation code are always in sync with each other. Users of your library will be including your header file, so your library implementation code in your .C files needs to do the same.

Building the Library

Building a library is the same as building an application through the Build menu. However, if you're in the habit of running your applications to trigger a build, keep in mind that a .LIB file is not an executable file. So, for example, if you click Debug | Start without debugging, your library project will build, but you'll get an error message say that Visual Studio can't start the .LIB file because it's not a valid application.

Remember that a library is not an executable. It is meant only for linking into applications that make use of the library. Your best bet is to get into the habit of using the Build menu to build your library.

After you have a static library created, applications that want to use this library will need access to both the .LIB library file and its associated .H header file. There are several ways to accomplish this, depending on how your project code is organized. The following sections cover linking to the library (a) if your application project is in the same Visual Studio solution as your library, and (b) if your application project is in a different solution from your library.

Linking the Library to an Application (Same Solution)

Let's say you have an application project, MyApplication, that needs to use your library, and that project is in the same Visual Studio solution as your library. Here is one approach to deal with this situation.

1. With your solution open so that you can see both your library project and your application project in Solution Explorer, right-click the application's project name (e.g., MyApplication) in Solution Explorer.
2. On the context menu, click **Add | References**.
3. In the **Property Pages** window, the left pane should show **References** selected under **Common Properties**. In the right pane, click **Add New Reference**.
4. In the **Add Reference** window, you should see your library project listed (e.g., MyFirstLibrary). Click to check the box next to your library, and click **OK**.
5. In the **Property Pages** window, navigate to **Configuration Properties | C/C++ | General**.
6. In the right pane of the **Property Pages** window, click the down-arrow next **to Additional Include Directories**, and click **<Edit>.**
7. In the **Additional Include Directories** window, click in the blank area, and click the ellipsis **...** button.
8. In the **Select Directory** window, navigate to the folder (directory) containing your library's header file. This will typically be up one level from your application's project folder, and then down one level to your library's project folder.

    - For example, if your solution contains two projects, MyApplication and MyFirstLibrary, you would need to climb out of the MyApplication project folder, and then climb into the MyFirstLibrary project folder.
9. Once you have chosen the folder that contains your library header file, click **Select Folder**.
10. In the **Additional Include Directories** window, click **OK**. At this point, you should see the pathname of the folder added in **Additional Include Directories** in the Project Properties right pane.
11. In the **Project Properties** window, click **OK**.

Now, in your application project, you can include the library's header file and when you build your application, it will link automatically to your library.

Linking the Library to an Application (Different Solution)

Let's say you have an application project that needs to use your library, but that application project is in a separate Visual Studio solution. Here is one approach to deal with this situation.

1. Choose a location where you want the .H and .LIB files to reside, so that other projects can access them. For this discussion, we'll choose C:\MyLibraries. Ensure that this folder exists. If it doesn't exist, create it.
2. In your library project, add a post-build event, which will copy files into that folder.
    a. In Visual Studio, select your library project in **Solution Explorer**.
    b. On the Project menu, click Properties.
    c. In the left pane of the **Property Pages** window, navigate to **Configuration Properties | Build Events | Post-Build Event**.
    d. In the right pane of the **Property Pages** window, click the box next to **Command Line**, click the down arrow at the right end of the box, and click **Edit**.
    e. Type the following lines into the **Command Line** window:

    ```
    copy "$(ProjectDir)\XXXXXXXX.h" C:\MyLibraries\
    copy "$(TargetPath)" C:\MyLibraries\
    ```

    where XXXXXXXX.h is the name of the header file you want to publish for use with your library. Substitute your own header file name for this, and be sure to include the .h extension. The first line copies the header file, and the second line copies the .LIB library file.
3. In the **Command Line** window, click **OK**.
4. In the **Property Pages** window, click **OK**.
5. Click **Build | Rebuild** to rebuild the library. After the build completes, you should see your .LIB file and your .H header file in the C:\MyLibraries folder.
6. Close your library solution.
7. Now, using Visual Studio, create or open the solution containing the application that wants to link to your library.
8. On the **Project** menu, click **Properties**.

9. In the left pane of the **Property Pages** window, navigate to **Configuration Properties | Linker | Input**.
10. In the right pane of the window, you should see an entry named **Additional Dependencies**, which contains a long list of .LIB library files. Click that box.
11. At the right end of that box, click the down-arrow and click **Edit**.
12. In the upper box of the **Additional Dependencies** window, as shown at the right, type the full path name of the library file on your computer, including its .LIB file extension. Then click **OK**.
13. In the **Property Pages** window, you should see that the library you just added appears in the **Additional Dependencies** box, followed by **;%(AdditionalDependencies)**. The entire entry should appear in bold, indicating that you have made a change to this property.
14. In the **Property Pages** window, click **OK**.
15. When your source code includes the header file associated with the library, specify the full path name to the header file.
16. Build your application project. The header file should be pulled in at compile time and the library file should be pulled in at link time.

Note that, depending on a number of factors, libraries may or may not be compatible between versions of Visual Studio, or between the same versions of Visual Studio if some of the configuration settings differ.