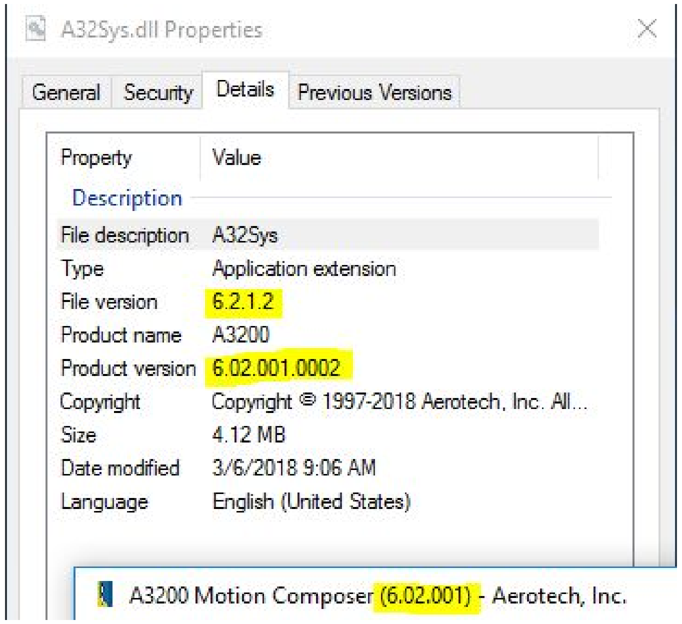
Building Projects Against the A3200 C Library in Visual Studio:

1. The version of the library DLLs must be in sync with each other and with the version of the A3200 controller installed



Property	Value
Description	
File description	A32Sys
Type	Application extension
File version	6.2.1.2
Product name	A3200
Product version	6.02.001.0002
Copyright	Copyright © 1997-2018 Aerotech, Inc. All...
Size	4.12 MB
Date modified	3/6/2018 9:06 AM
Language	English (United States)

A3200 Motion Composer (6.02.001) - Aerotech, Inc.

- The [InstallDir]\CLibrary\bin\ and [InstallDir]\CLibrary\bin64\ directories must be added to the PATH or their contents must be copied and pasted into the Output Directory of any C application that uses the A3200 C Library
- The linker must be setup to find the A3200 C Library (A3200C.lib and A3200C64.lib as Additional Dependencies)
- The [InstallDir]\CLibrary\include\ directory must be added as an Additional Include Directory

Sample Projects and Code Snippets:

- There are multiple sample projects located in the Samples directory of the installation directory (e.g. C:\Program Files\Aerotech\A3200\Samples)
    - These are also useful for determining the proper Visual Studio project properties
- There are example usages of some of the functions in their A3200 Programming Help file topics
- The FAQs and example application in this guide are a good starting point

Using the A3200 C Library:

- Add the line “#include “A3200.h” to use the A3200 C Library
- Unless indicated otherwise, the functions in the A3200 C Library are blocking (the next program line is not executed until the current one completes)
    - Library methods that are wrappers for AeroBasic commands have the same blocking behavior as the AeroBasic command
- The A3200 C Library is not meant for real-time motion control and any commands issued through the library will have a latency of anywhere from 5-10ms
- Commands that are executed from multiple threads must be synchronized using something like locks, mutexes, semaphores, etc.
- A3200 C Library functions will return FALSE if an error occurs
    - Call A3200GetLastError() to get more information on the error

Frequently Asked Questions:

1. *Will my C application work with a version of the A3200 software that is different than the version of the A3200 C Library that I used to build my application?*

Small differences in version might mean that your application code is source-level compatible between the different versions of the A3200 C Library, but **users will still need to rebuild their application against the version of the A3200 C Library that matches the A3200 controller version that they intend to use with their application**. Follow this procedure when updating C applications to function properly with a different controller version:

I. Check the release notes to see what has changed between versions of the A3200 C Library and if upgrading a project from 2.x, please see the "Upgrading Programs From 2.x" topic in the A3200 Programming Help file, as well as any "Related Topics"
II. If there have been updates to a feature that was used in or application, update your source code to use the correct syntax and to use the features of the controller properly
III. Copy all of the files in the intend target A3200 controller versions' [InstallDir]\C\bin and [InstallDir]\C\bin64 directories and paste them in the application's Output Directory
IV. Add the [InstallDir]\CLibrary\include\ directory as an Additional Include Directory
V. Add A3200C.lib and A3200C64.lib as Additional Dependencies in the Linker properties
VI. Rebuild the application

2. *What is the recommended approach for handling errors when using the A3200 C Library?*

There is not one correct approach to handling errors that occur when using the A3200 C Library. Most all of the functions in the A3200 C Library will return FALSE on error and the A3200GetLastError() or A3200GetLastErrorString() function can be used to retrieve the last error that occured. The C applications in the Samples directory setup an error printing function that is called when any of the A3200 C Library function calls errors (cleanup code is also used and is shown below).

```
// This function will print whatever the latest error was
void PrintError();

int main(int argc, char **argv)
{
	// Handle to give access to A3200
	A3200Handle handle = NULL;

	printf("Connecting to A3200. Initializing if necessary.\n");
	if (!A3200Connect(&handle)) { PrintError(); goto cleanup; }

		// application code

cleanup:
	if(handle != NULL) {
		if(!A3200MotionDisable(handle, TASKID_Library, axisMask)) { PrintError(); }
		if(!A3200Disconnect(handle)) { PrintError(); }
	}

	return 0;
}

void PrintError() {
	CHAR data[1024];
	A3200GetLastErrorString(data, 1024);
	printf("Error : %s\n", data);
}
```

3. *How do I edit controller parameters?*

The **Parameter Functions** allow users to manipulate the active controller parameters as well as the parameters in an A3200 parameter file. The controller will revert back to the parameter values in the active parameter files' on controller reset.

Edit parameters on the SMC:

```
if (!A3200ParameterSetValue(handle, PARAMETERID_HomeOffset, AXISINDEX_00, 75)) { PrintError(); goto cleanup; }
if (!A3200ParameterSetValueString(handle, PARAMETERID_UserString0, NULL, "Hello World")) { PrintError(); goto cleanup; }
```

Open, manipulate, and save a parameter file:

```
A3200ParameterFile params = NULL;
CHAR paramsPath[MAX_PATH] = "C:\\Users\\Public\\Documents\\Aerotech\\A3200\\User Files\\myParams.prma";

if (!A3200ParameterFileOpen(paramsPath, &params)) { PrintError(); goto cleanup; }
if (!A3200ParameterFileSetValue(handle, params, PARAMETERID_HomeOffset, AXISINDEX_00, 50)) { PrintError(); goto cleanup; }
if (!A3200ParameterFileSetValueString(handle, params, PARAMETERID_UserString0, NULL, "Hello World pt. 2")) { PrintError(); goto cleanup; }
if (!A3200ParameterFileSave(params, paramsPath)) { PrintError(); goto cleanup; }
if (!A3200ParameterFileClose(params)) { PrintError(); goto cleanup; }
```

4. *How do I read and write controller variables?*

The **Variable Functions** can be used to retrieve and set the value of controller variables. To access a Program Variable, the TASKID argument should be the task of the running program that defined the Program Variable (DVAR). These functions can also be used for fieldbus variables.

```
double taskVars[5] = { 0, 1, 2, 3, 4 };
double globalVars[5];
CHAR globalString[128];

if (!A3200VariableSetGlobalDouble(handle, 0, 23.32)) { PrintError(); goto cleanup; }
if (!A3200VariableSetTaskDoubles(handle, 0, 0, taskVars, 5)) { PrintError(); goto cleanup; }
if (!A3200VariableGetGlobalDoubles(handle, 0, globalVars, 5)) { PrintError(); goto cleanup; }
if (!A3200VariableSetValueStringByName(handle, TASKID_02, "$strtask[3]", "Hello world")) { PrintError(); goto cleanup; }
if (!A3200VariableSetValueByName(handle, 0, "$global[2]", 1.23)) { PrintError(); goto cleanup; }
if (!A3200VariableGetValueStringByName(handle, 0, "$strglob[1]", globalString, 128)) { PrintError(); goto cleanup; }
if (!A3200VariableSetValueByName(handle, TASKID_01, "$testDVAR", 4.56)) { PrintError(); goto cleanup; }
```

5. *How do I read and write Modbus variables?*

The Modbus Functions are based on the old Modbus implementation and the new Fieldbus Mapping Dialog implementation should be used (see the Fieldbus Mapping Dialog topic in the A3200 Help file) with the **A3200VariableGetValueByName()** and **A3200VariableSetValueByName() functions** with a **TASKID argument of 0**.

```
DOUBLE modbusOWordStatus;
DOUBLE modbusIBit;

if (!A3200VariableSetValueByName(handle, 0, "$MyModbusOWord", 32)) { PrintError(); goto cleanup; }
if (!A3200VariableGetValueByName(handle, 0, "$MyModbusOWordStatus", &modbusOWordStatus)) { PrintError(); goto cleanup; }
if (!A3200VariableGetValueByName(handle, 0, "$MyModbusIBit", &modbusIBit)) { PrintError(); goto cleanup; }
```

6. *Is the A3200 C Library thread safe?*

The A3200 C Library is **not** thread safe. If an application uses threading, all A3200 function calls must be manually synchronized (with mutexes, locks, semaphores, etc.) or the controller should only be accessed by a single thread.

7. *Which A3200 C Library functions are blocking?*

Unless indicated otherwise, the A3200 .NET Library methods have the same blocking behavior as their AeroBasic equivalents. Please see the "**Blocking Behavior of the Libraries**" topic in the A3200 Programming Help file for more information.

If a user would like to get around the blocking behavior of a function, they can place a controller task in **Queue Mode** and add the command to the task's execution queue (see the "Queue Mode Overview" topic in the A3200 Help file for more information). There is an example the shows how to use a task's Queue Mode in the A3200 Programming Help file documentation for the A3200ProgramInitializeQueue() function.

Another option is to write an AeroBasic program that contains the desired commands and use the **Program/Task Control Functions** to load, start, and stop the program. **The A3200ProgramRun() function does not block.** Use the A3200ProgramGetTaskState() function to check for program completion.

8. *How do I modify my A3200 configuration using the A3200 C Library?*

Use the **Configuration Functions** to modify the A3200 configuration. The changes will not be applied until after the controller is started or reset (controller must be reset if it was already running).

```
A3200ConfigurationHandle configHandle = NULL;

A3200ProgramAutomationFile myInclude = { "C:\\Users\\Public\\Documents\\Aerotech\\A3200\\User Files\\Include.pgm", PROGRAMAUTOMATIONMODE_Include, TASKMASK_None };
A3200ProgramAutomationFile myTransform = { "C:\\Users\\Public\\Documents\\Aerotech\\A3200\\User Files\\Transform.pgm", PROGRAMAUTOMATIONMODE_Run, TASKMASK_02 };

CHAR galvoCalPath[MAX_PATH] = "C:\\Users\\Public\\Documents\\Aerotech\\A3200\\User Files\\MyGalvo.cal";

if (!A3200ConfigurationOpen(&configHandle)) { PrintError(); goto cleanup; }
if (!A3200ConfigurationProgramAutomationAdd(configHandle, myInclude)) { PrintError(); goto cleanup; }
if (!A3200ConfigurationProgramAutomationAdd(configHandle, myTransform)) { PrintError(); goto cleanup; }
if (!A3200ConfigurationCalibrationFileSet(configHandle, AXISCALIBRATION_FILETYPE_GALVO_2D, galvoCalPath)) { PrintError(); goto cleanup; }
if (!A3200ConfigurationSave(configHandle)) { PrintError(); goto cleanup; }
if (!A3200ConfigurationClose(configHandle)) { PrintError(); goto cleanup; }
```

9. *I can't find the _______ command in the A3200 C Library. How do I execute a command that isn't part of the API?*

Users can issue commands that are not offered in the A3200 C Library by using the **A3200CommandExecute() function**. Pass a string that contains one or more AeroBasic command(s) delimited by \n (a newline) and the command(s) will be compiled and executed as immediate commands. The returnValue argument should be NULL if the the AeroBasic string is not expected to return a value. Otherwise, an error will occur. The **Program/Task Control Functions** can also be used to run AeroBasic programs.

```
A3200Handle handle = NULL;
DOUBLE ampTemp;

if (!A3200Connect(&handle)) { PrintError(); goto cleanup; }
if (!A3200CommandExecute(handle, TASKID_01, "DRIVEINFO(X, DRIVEINFO_AmplifierTemperature)", &ampTemp)) { PrintError(); goto cleanup; }
if (!A3200CommandExecute(handle, TASKID_01, "ENABLE X\nLINEAR X 10\nENABLE Y\nLINEAR Y 20", NULL)) { PrintError(); goto cleanup; }
```

10. *How can I call an AeroBasic subroutine from within my .NET application?*

Add the program that contains the subroutine to **Program Automation as Download** in A3200 Configuration Manager, or use the **A3200ProgramLoad() function** to load the .PGM to the controller. The **A3200CommandExecute() function** can be used to call the subroutine from a specific task.

```
A3200Handle handle = NULL;
CHAR ProgramFileLocation[MAX_PATH] = "C:\\Users\\Public\\Documents\\Aerotech\\A3200\\User Files\\myProgram.pgm";

if (!A3200Connect(&handle)) { PrintError(); goto cleanup; }
if (!A3200ProgramLoad(handle, TASKID_02, ProgramFileLocation)) { PrintError(); goto cleanup; }
if (!A3200VariableSetValueStringByName(handle, TASKID_Library, "$strglob[0]", "myProgram.pgm")) { PrintError(); goto cleanup; }
if (!A3200VariableSetValueStringByName(handle, TASKID_Library, "$strglob[1]", "myFunction")) { PrintError(); goto cleanup; }
if (!A3200CommandExecute(handle, TASKID_01, "FARCALL strglob[0] strglob[1]", NULL)) { PrintError(); goto cleanup; }
```

An ONGOSUB can also be set up in an AeroBasic program to execute the block of code upon some event, like a Task Error. This program containing the ONGOSUB would need to be run via the A3200ProgramLoad() then A3200ProgramStart() functions or A3200ProgramRun() function, or added to Program Automation as Run or RunSilent.

11. *I can't find VELOCITY ON, PVT, or their equivalent functions in the A3200 C Library. How do I command a series of moves with velocity profiling and/or precise timing in my C application?*

There are not APIs for velocity profiling or PVT motion because the A3200 libraries are not intended for real-time motion control. Users could see between 5-10ms of latency between immediate commands and the controller is not able to effectively plan motion to achieve velocity blending throughout a series of commanded moves, *even after a user issues A3200CommandExecute(handle, TASKID_XX, "VELOCITY ON", NULL)*.

12. *What are some best practices when using the Program/Task Control Functions?*

- When possible, use .PGM files instead of .OGM files. Errors can occur when an .OGM is used without the .PGM present.
- Check return values and retrieve the error information when the functions return FALSE
- Use END PROGRAM to end AeroBasic programs
    - Otherwise, a program may not complete smoothly when using Buffered Run
- Use A3200ProgramGetTaskState() or A3200ProgramGetTaskStateString() to check a task's state before running programs in it
    - For example, the A3200ProgramStart() function should only be used when the program has been downloaded and associated with the task and when the task state is: *ProgramComplete*, *ProgramReady*, or *ProgramPaused*

- There are examples for:
    - A3200ProgramBufferedRun() in the Buffered Run Queue example in the Samples directory
    - A3200ProgramInitializeQueue() in the A3200 Programming Help file documentation for the function
    - A3200ProgramRun() and A3200ProgramStopAndWait() in the Console Example in the Samples directory

13. Can the A3200 C Library be used to control a Hexapod?

With the Hexapod files setup correctly with Program Automation, the **A3200CommandExecute() function** can be used to issue Hexapod commands to do things like setup and/or activate tools and setup coordinate systems.

The Hexapod transformations will be consuming motion commanded to the virtual axes in real-time, regardless of the source of the commands. Point to point moves can be commanded through the API without issue. If velocity profiling is desired, **Queue Mode** can be used or the **A3200ProgramRun() function** can be used to run AeroBasic programs that contain the VELOCITY ON command followed by the sequence of moves (See FAQs #11 and #12).

14. What steps are required to use a C application on a client PC to control a remote A3200 in a Remote Server configuration?

After purchasing the Remote option, install the A3200 client-only software on the client PC and follow the “Remote Server and Client Installation and Configuration” A3200 Help file topic. Make sure the the programs, calibration, and parameter files are all on the client PC. There shouldn’t be anything on the server PC (including A3200 Program Automation). Use the A3200 and the C Library on the client PC as you normally would and A3200 will automatically take care of communicating the information over TCP/IP.

15. How do I poll for diagnostics and status items through the A3200 C Library?

I. The **Status Commands** can be used to retrieve various status items and the functions work similar to the AXISSTATUS, TASKSTATUS, and SYSTEMSTATUS AeroBasic commands. *The A3200 Programming Help file documentation for the A3200GetStatusItems() function contains an example with code to retrieve multiple status values of a different kind at the same time.*

II. The **Data Collection Functions** can be used to retrieve a specific set of diagnostic items for a finite or infinite number of samples. Users can configure their data collection (samples, signals, etc.), start/stop the data collection, check the status of their collection, and retrieve the sample points from their data collection (data collection signals will be in counts). *There are a few examples of data collections in the A3200 Programming Help file documentation for the Data Collection Functions.*

III. The **A3200CommandExecute() function** can be used to issue AeroBasic status commands as immediate command strings:

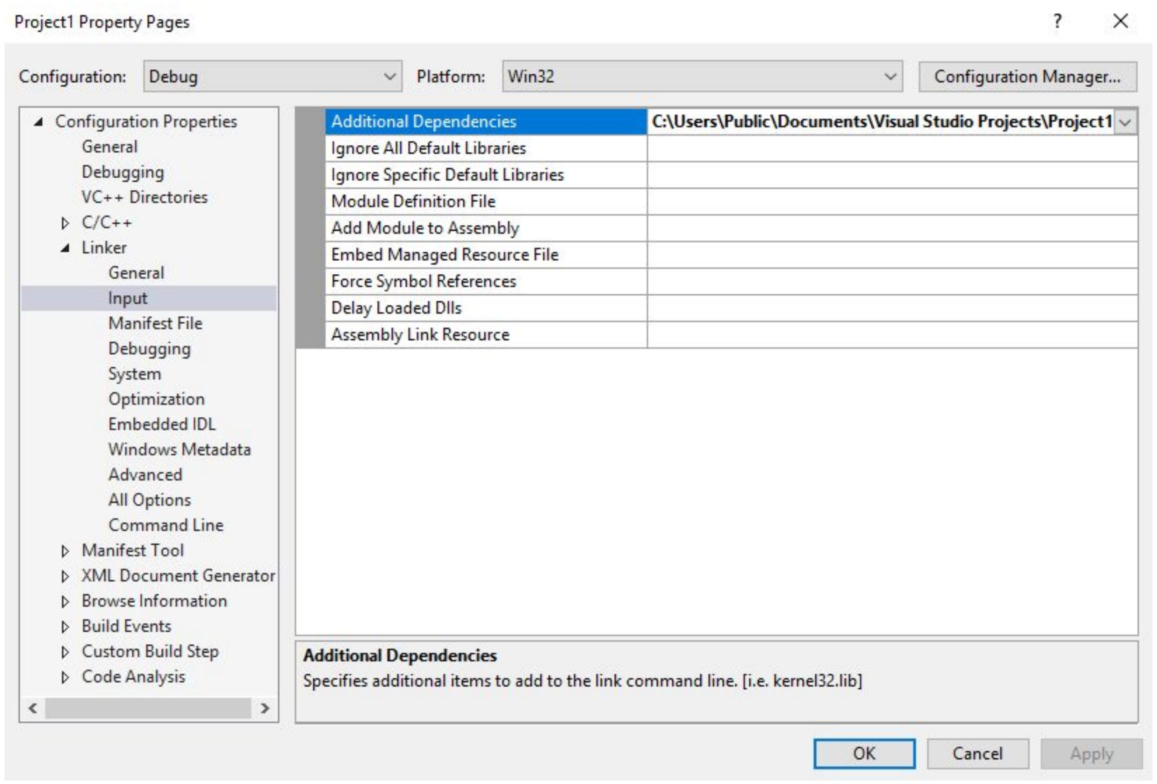
```
A3200Handle handle = NULL;
DOUBLE coordinatedSpeedTarget;

if (!A3200Connect(&handle)) { PrintError(); goto cleanup; }
if (!A3200CommandExecute(handle, TASKID_01, "TASKSTATUS($taskindex, DATAITEM_CoordinatedSpeedTarget)",
&coordinatedSpeedTarget)) { PrintError(); goto cleanup; }
```

Troubleshooting Tips:

This section contains some suggestions for troubleshooting C applications when using Visual Studio and targeting the A3200 C Library

When I try to build my application, I am seeing linker errors:

Make sure that the full paths to A3200C.lib and A3200C64.lib are provided as Additional Dependencies in the Project Properties:



Configuration: Debug
Platform: Win32

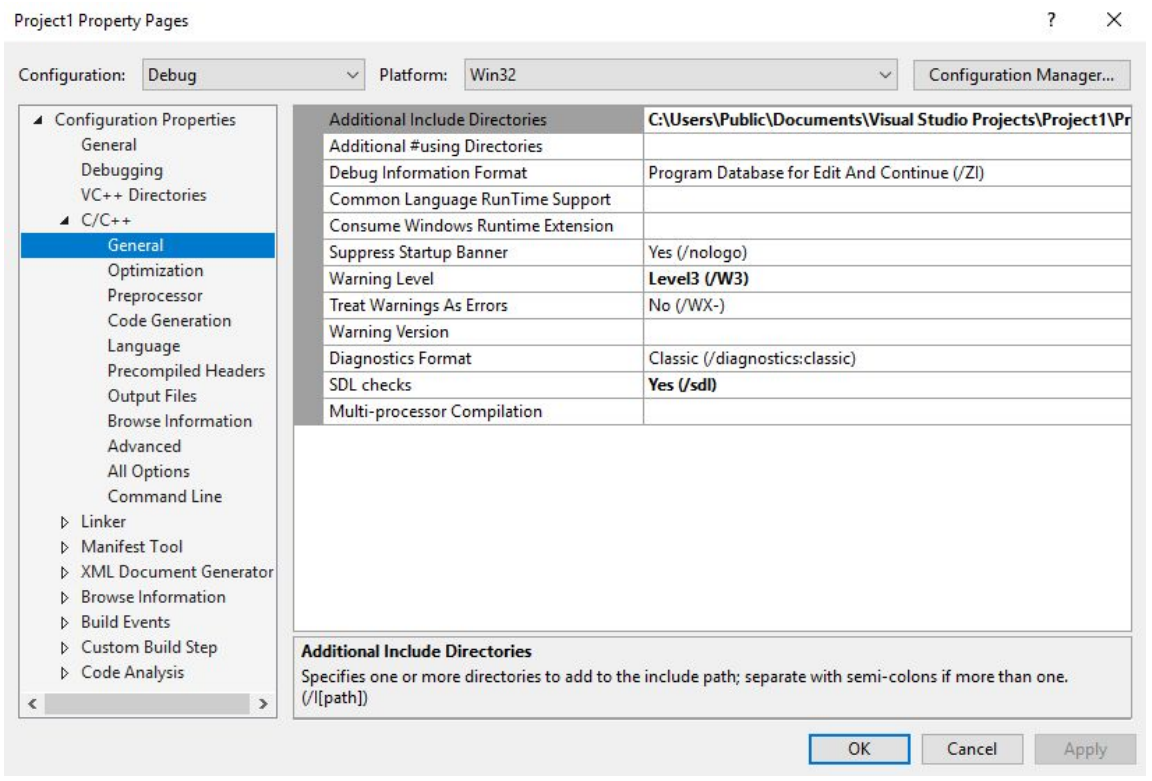
Additional Dependencies	C:\Users\Public\Documents\Visual Studio Projects\Project1
Ignore All Default Libraries	
Ignore Specific Default Libraries	
Module Definition File	
Add Module to Assembly	
Embed Managed Resource File	
Force Symbol References	
Delay Loaded Dlls	
Assembly Link Resource	

Additional Dependencies
Specifies additional items to add to the link command line. [i.e. kernel32.lib]

When I try to build my application, I am seeing compiler errors:

Make sure that the line **#include A3200.h** is added to the top of any program that uses the the A3200 C Library.

Also, confirm that the path to the include directory that contains the header files is added to the Project Properties as Additional Include Directories:



Additional Include Directories	**C:\Users\Public\Documents\Visual Studio Projects\Project1\Pr**
Additional #using Directories	
Debug Information Format	Program Database for Edit And Continue (/ZI)
Common Language RunTime Support	
Consume Windows Runtime Extension	
Suppress Startup Banner	Yes (/nologo)
Warning Level	**Level3 (/W3)**
Treat Warnings As Errors	No (/WX-)
Warning Version	
Diagnostics Format	Classic (/diagnostics:classic)
SDL checks	**Yes (/sdl)**
Multi-processor Compilation	

Additional Include Directories
Specifies one or more directories to add to the include path; separate with semi-colons if more than one. (/I[path])

When I try to run my application, I am seeing runtime errors that say that I am missing DLLs:

Add [InstallDir]\CLibrary\bin\ and [InstallDir]\CLibrary\bin64\ to your PATH or copy all of the DLLs within the bin and bin64 and paste them in the project's output directory.

I followed all of the instructions in this setting and I am still having trouble compiling and running my application:

Run one of the sample A3200 C Library applications (found in [InstallDir]\Samples\C\CLibrary). If this builds and runs, try:

1. Copying the source from the example project and trying to run it within the broken project
2. Looking at example's Project Properties and making sure that the broken project's properties match any of the bold fields (which are different values than the default)

Some A3200 C Library functions will error when passing certain values as arguments:

If it is implied that the argument is conceptually an integer, please ensure that the value does not have any fractional part. Some examples might be arguments like (NumElements, StartIndex, NumCycles).

Changes to A3200 Configuration using the Configuration Functions are not taking effect when I connect to the controller using the A3200Connect() function:

If the A3200 is already running, it must be reset before changes to configuration will be active. If the problem persists, ensure that any paths passed to the Configuration Functions have double backslashes (\\). *For more information, see: https://docs.microsoft.com/en-us/cpp/c-language/escape-sequences*

NOTE: Galvo calibration cannot be applied when there is not a physical galvo controller connected. Remove any calibration using the Configuration Manager before developing/testing applications when there is not physical hardware connected.

Sometimes A3200 C Library functions return incorrect values or intermittently fail:

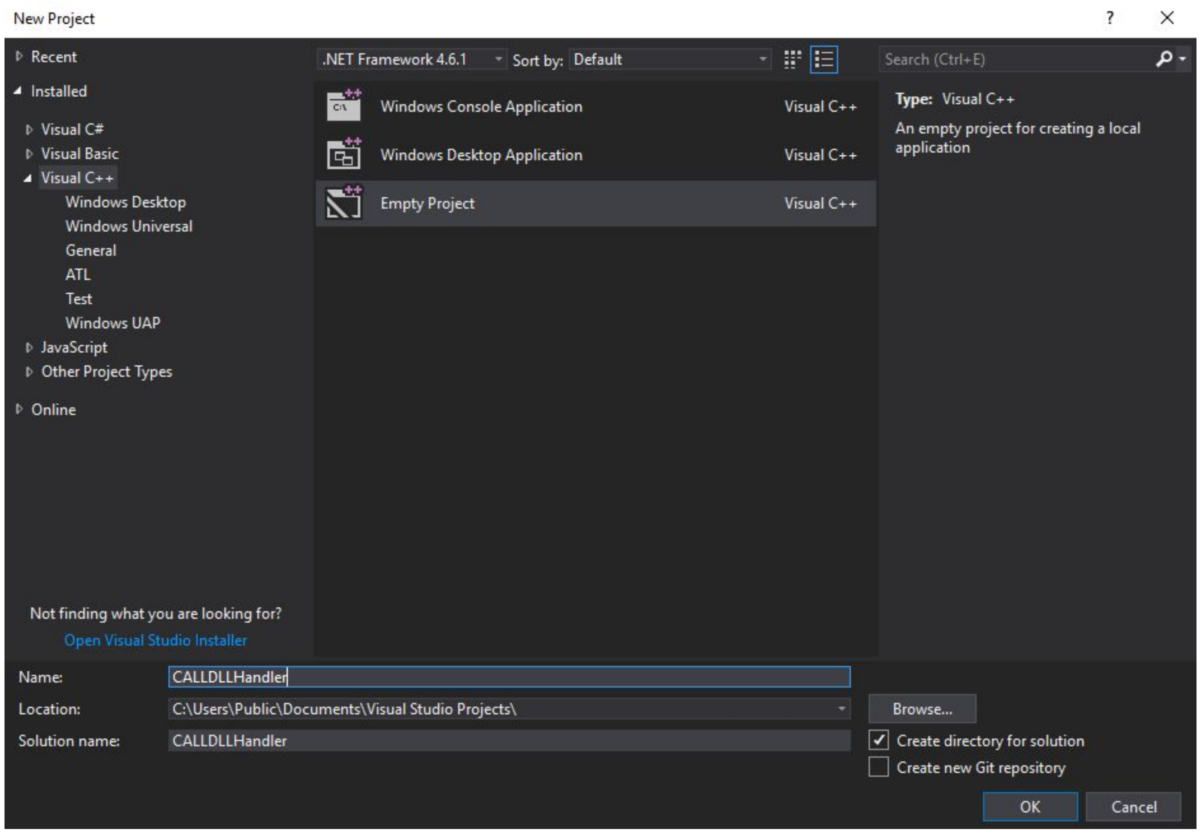
If the threading is used in the application, make sure that all controller accesses are synchronized (via locks, mutexes, semaphores, etc.) or only access the controller from a single thread.

Make sure that the A3200 C Library DLL version matches the A3200 software version being used. *See the "Building Projects Against the A3200 C Library in Visual Studio" section of this guide.*

Writing Custom CALLDLL Handling Functions with the A3200 C Library in Visual Studio:

The content in this section follows the "Writing Custom CALLDLL Handling Functions with the C Library" A3200 Help file topic. Visual Studio 2017 was used. The CALLDLL command can only call a function that is in a native Win32DLL. Other DLLs or COM objection can be called from within the the Win32 DLL. Aerotech does not support C++/CLR DLLs.

1. Create a New Project in Visual Studio. Select Visual C++ -> Empty Project



Type: Visual C++

An empty project for creating a local application

Name: CALLDLLHandler

Location: C:\Users\Public\Documents\Visual Studio Projects\

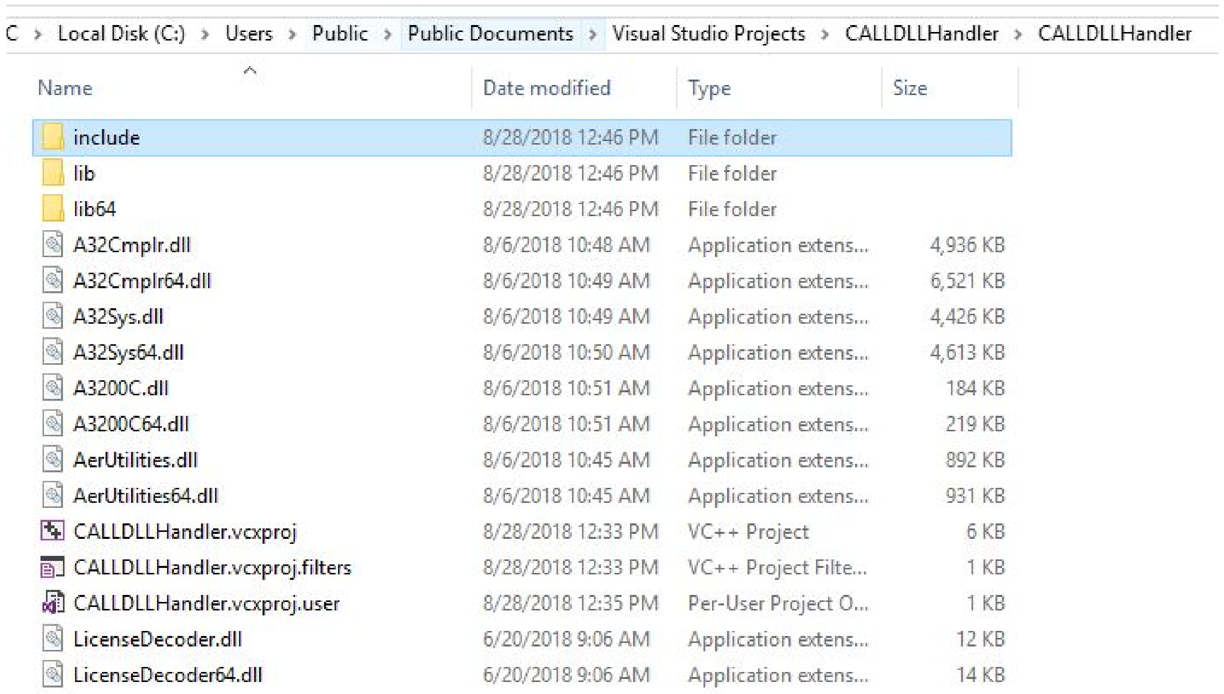
Solution name: CALLDLLHandler

[✓] Create directory for solution

[ ] Create new Git repository

2. Copy the contents of the [InstallDir]\CLibrary directory and paste into the project’s output directory

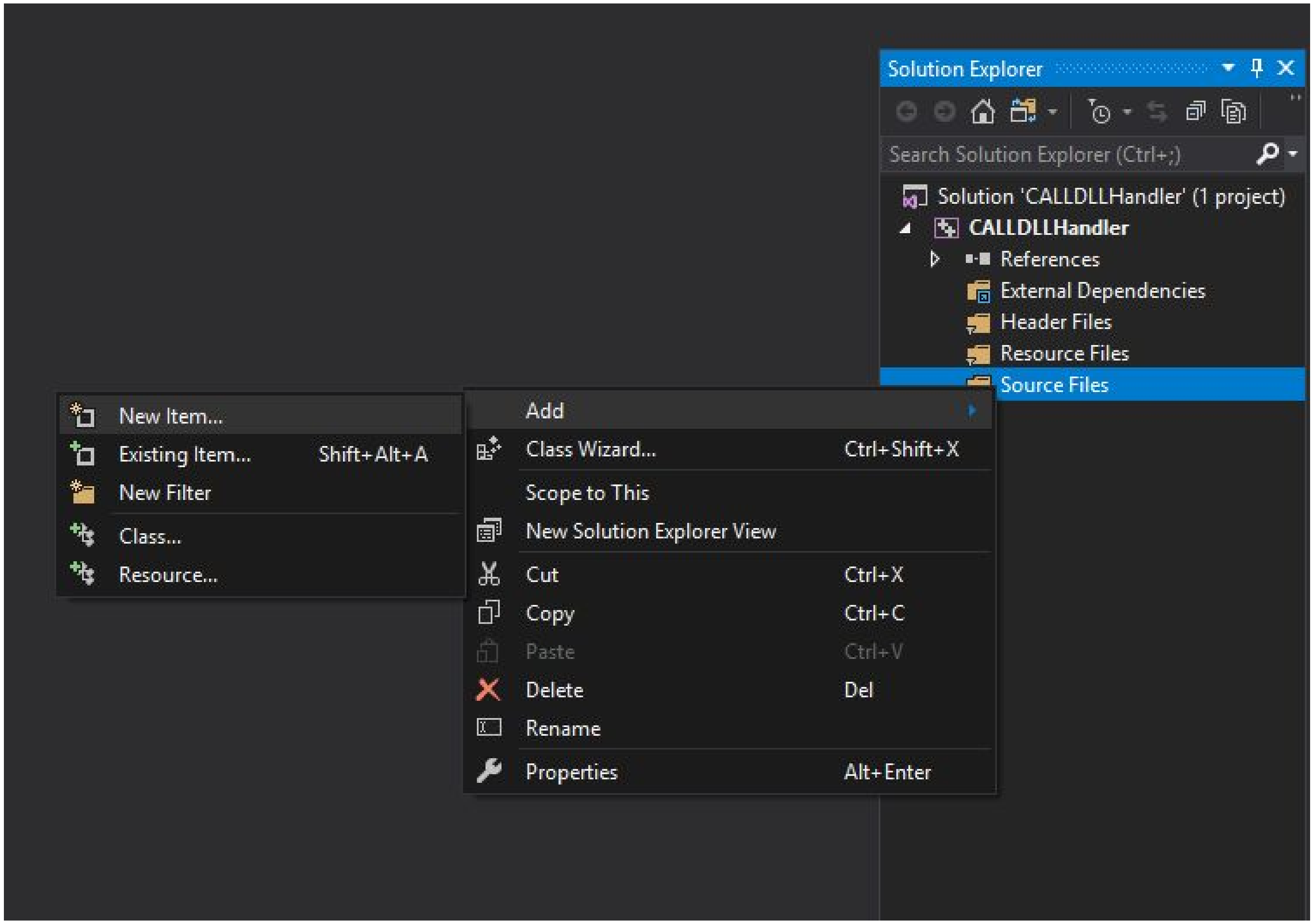
3. Copy the *.dll files from the bin and bin64 directories and paste them into the project’s output directory

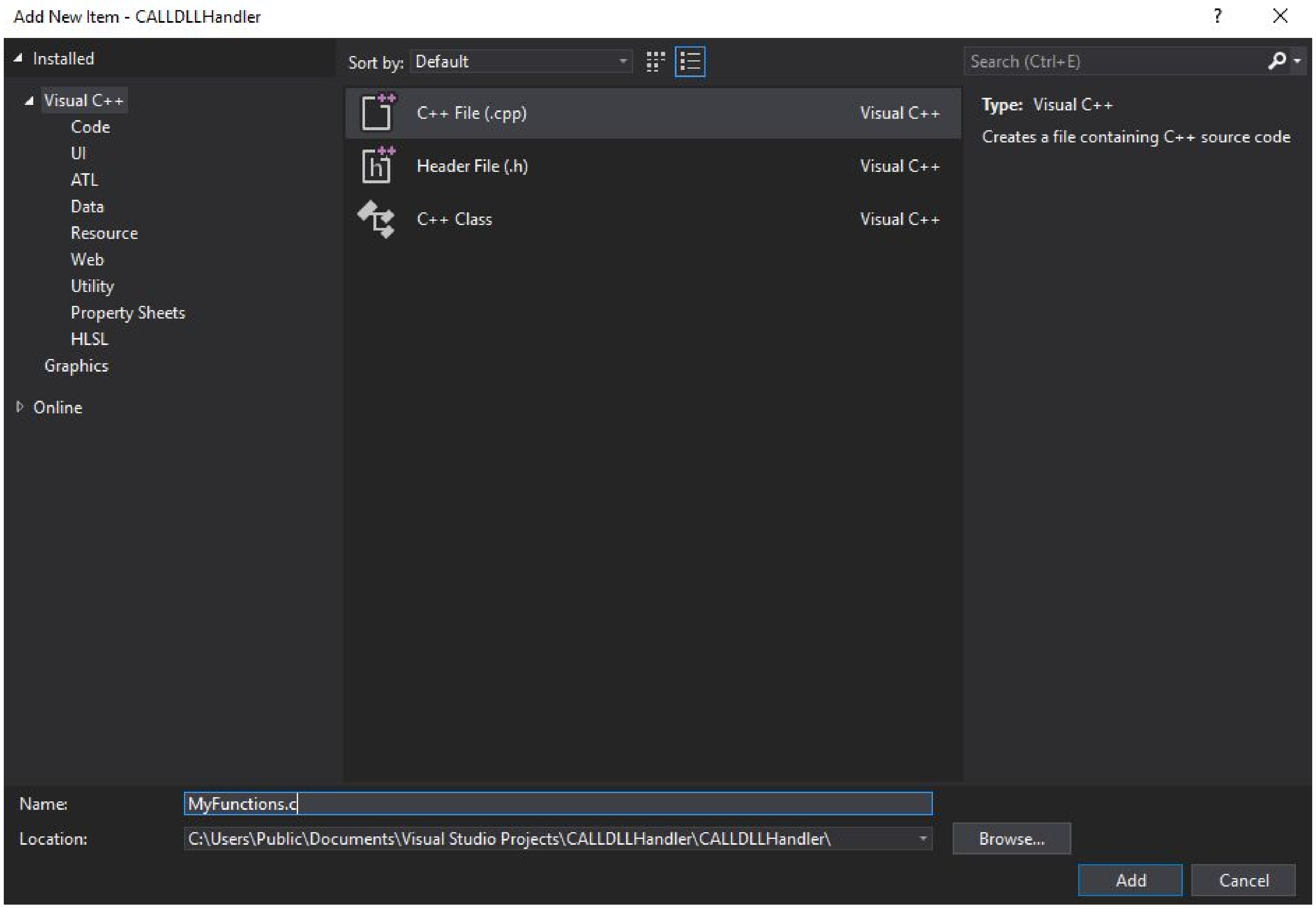


C > Local Disk (C:) > Users > Public > Public Documents > Visual Studio Projects > CALLDLLHandler > CALLDLLHandler

Name	Date modified	Type	Size
include	8/28/2018 12:46 PM	File folder	
lib	8/28/2018 12:46 PM	File folder	
lib64	8/28/2018 12:46 PM	File folder	
A32Cmplr.dll	8/6/2018 10:48 AM	Application extens...	4,936 KB
A32Cmplr64.dll	8/6/2018 10:49 AM	Application extens...	6,521 KB
A32Sys.dll	8/6/2018 10:49 AM	Application extens...	4,426 KB
A32Sys64.dll	8/6/2018 10:50 AM	Application extens...	4,613 KB
A3200C.dll	8/6/2018 10:51 AM	Application extens...	184 KB
A3200C64.dll	8/6/2018 10:51 AM	Application extens...	219 KB
AerUtilities.dll	8/6/2018 10:45 AM	Application extens...	892 KB
AerUtilities64.dll	8/6/2018 10:45 AM	Application extens...	931 KB
CALLDLLHandler.vcxproj	8/28/2018 12:33 PM	VC++ Project	6 KB
CALLDLLHandler.vcxproj.filters	8/28/2018 12:33 PM	VC++ Project Filte...	1 KB
CALLDLLHandler.vcxproj.user	8/28/2018 12:35 PM	Per-User Project O...	1 KB
LicenseDecoder.dll	6/20/2018 9:06 AM	Application extens...	12 KB
LicenseDecoder64.dll	6/20/2018 9:06 AM	Application extens...	14 KB

4. Add a .c file to the project (the extension will need to be changed from .cpp to .c)





C++ File (.cpp)	Visual C++
Header File (.h)	Visual C++
C++ Class	Visual C++

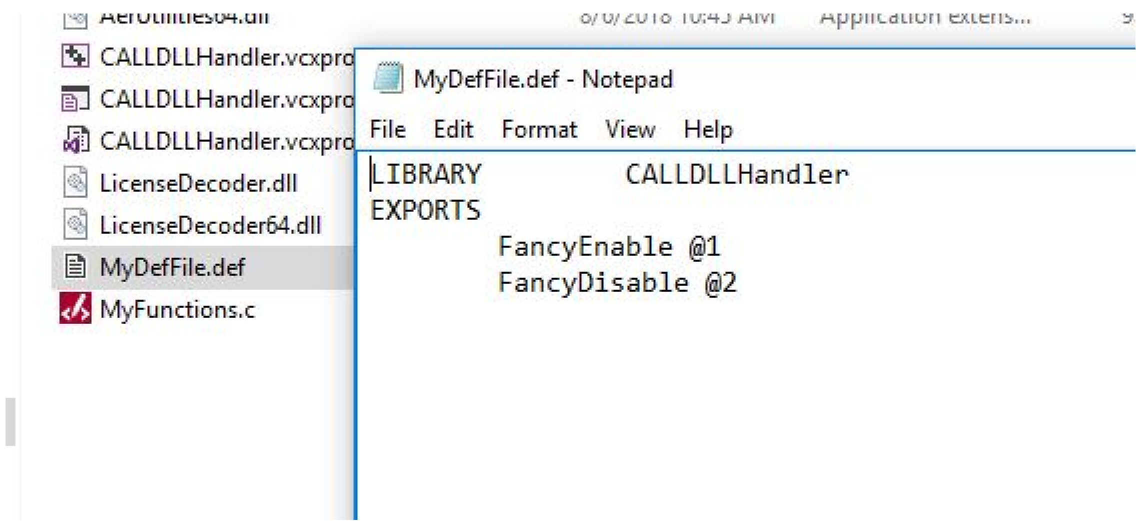
Type: Visual C++

Creates a file containing C++ source code

Name: MyFunctions.c

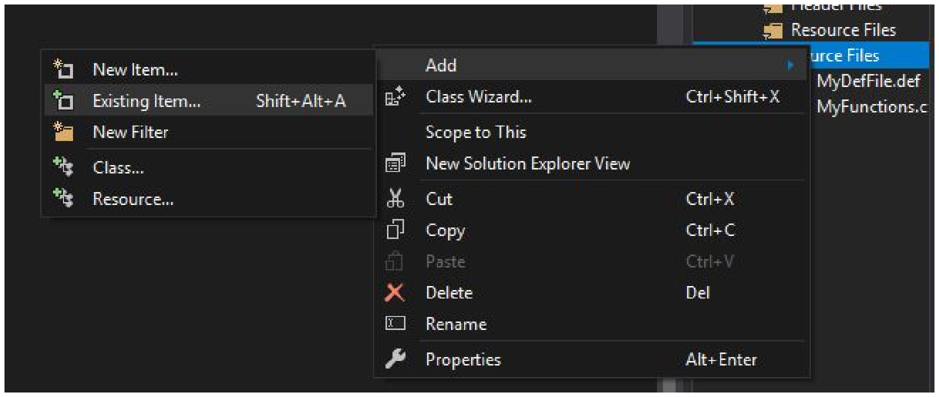
Location: C:\Users\Public\Documents\Visual Studio Projects\CALLDLLHandler\CALLDLLHandler\

5. Create a .txt file in the project's output directory, define the desired functions, and save with a .def extension

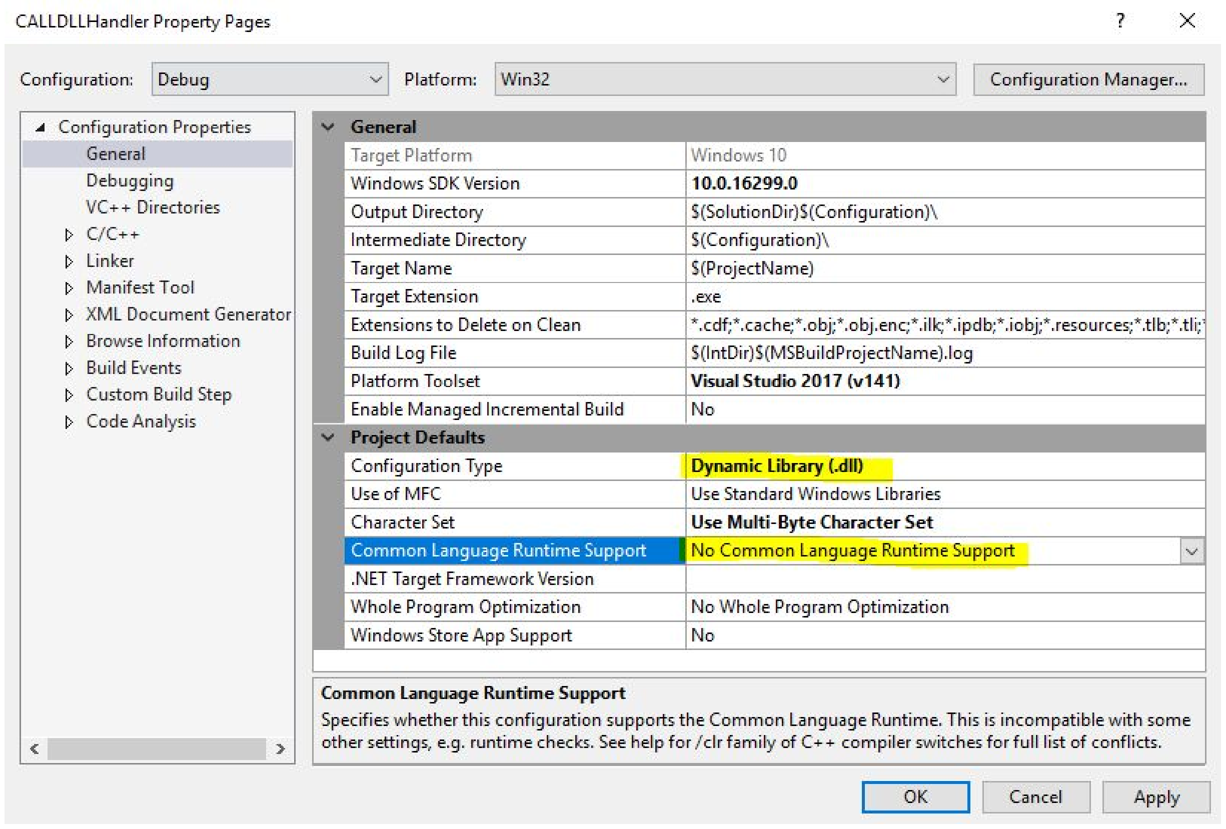


```
LIBRARY         CALLDLLHandler
EXPORTS
        FancyEnable @1
        FancyDisable @2
```

6. In the Solution Explorer, right-click on Source Files and click Add -> Existing Item... and select the *.def file



7. Open the Project Properties and apply the following changes:
   I. Under "General", change **Configuration Type** to **Dynamic Library (.dll)**
   II. Under "General", change **Common Language Runtime Support** to **No Common Language Runtime Support**



CALLDLLHandler Property Pages

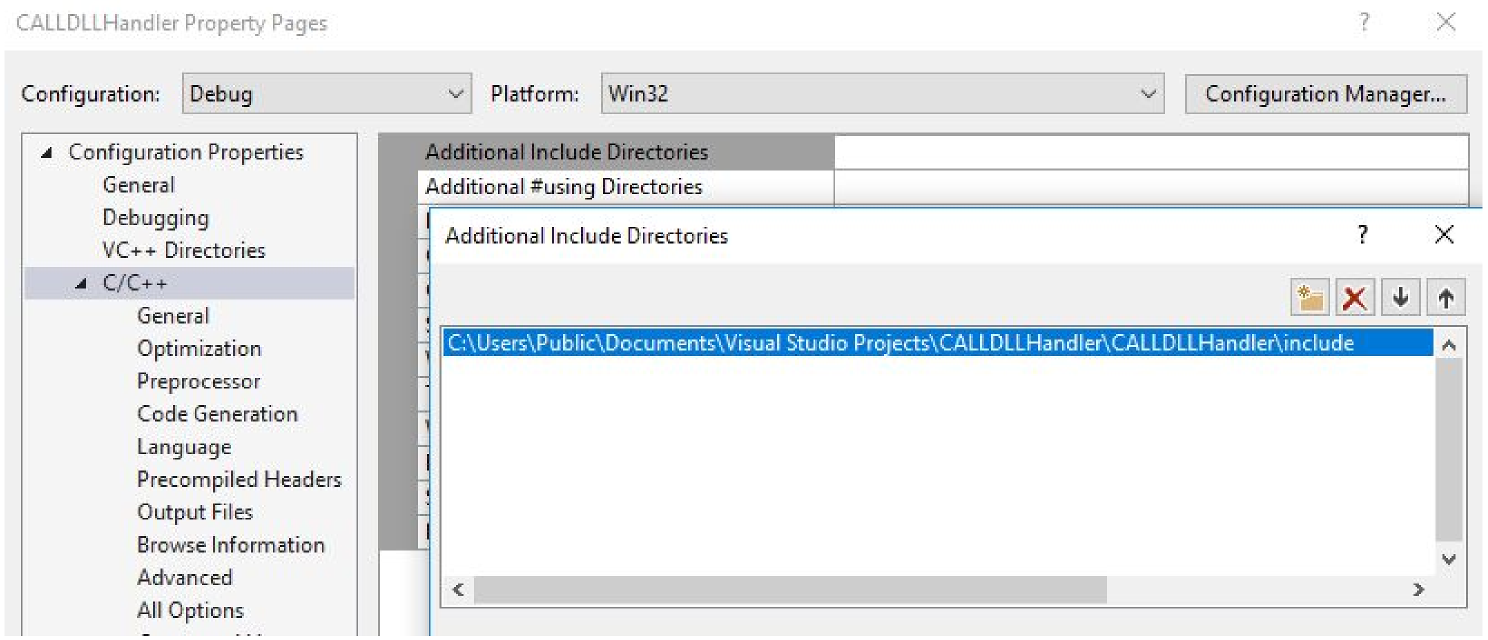
Configuration: Debug — Platform: Win32 — Configuration Manager...

Configuration Properties: General, Debugging, VC++ Directories, C/C++, Linker, Manifest Tool, XML Document Generator, Browse Information, Build Events, Custom Build Step, Code Analysis

General	
Target Platform	Windows 10
Windows SDK Version	**10.0.16299.0**
Output Directory	$(SolutionDir)$(Configuration)\
Intermediate Directory	$(Configuration)\
Target Name	$(ProjectName)
Target Extension	.exe
Extensions to Delete on Clean	*.cdf;*.cache;*.obj;*.obj.enc;*.ilk;*.ipdb;*.iobj;*.resources;*.tlb;*.tli;
Build Log File	$(IntDir)$(MSBuildProjectName).log
Platform Toolset	**Visual Studio 2017 (v141)**
Enable Managed Incremental Build	No
Project Defaults	
Configuration Type	**Dynamic Library (.dll)**
Use of MFC	Use Standard Windows Libraries
Character Set	**Use Multi-Byte Character Set**
Common Language Runtime Support	No Common Language Runtime Support
.NET Target Framework Version	
Whole Program Optimization	No Whole Program Optimization
Windows Store App Support	No

Common Language Runtime Support
Specifies whether this configuration supports the Common Language Runtime. This is incompatible with some other settings, e.g. runtime checks. See help for /clr family of C++ compiler switches for full list of conflicts.

III. Under “C/C++”, add the path to the include directory (from step 2.) as an **Additional Include Directory**

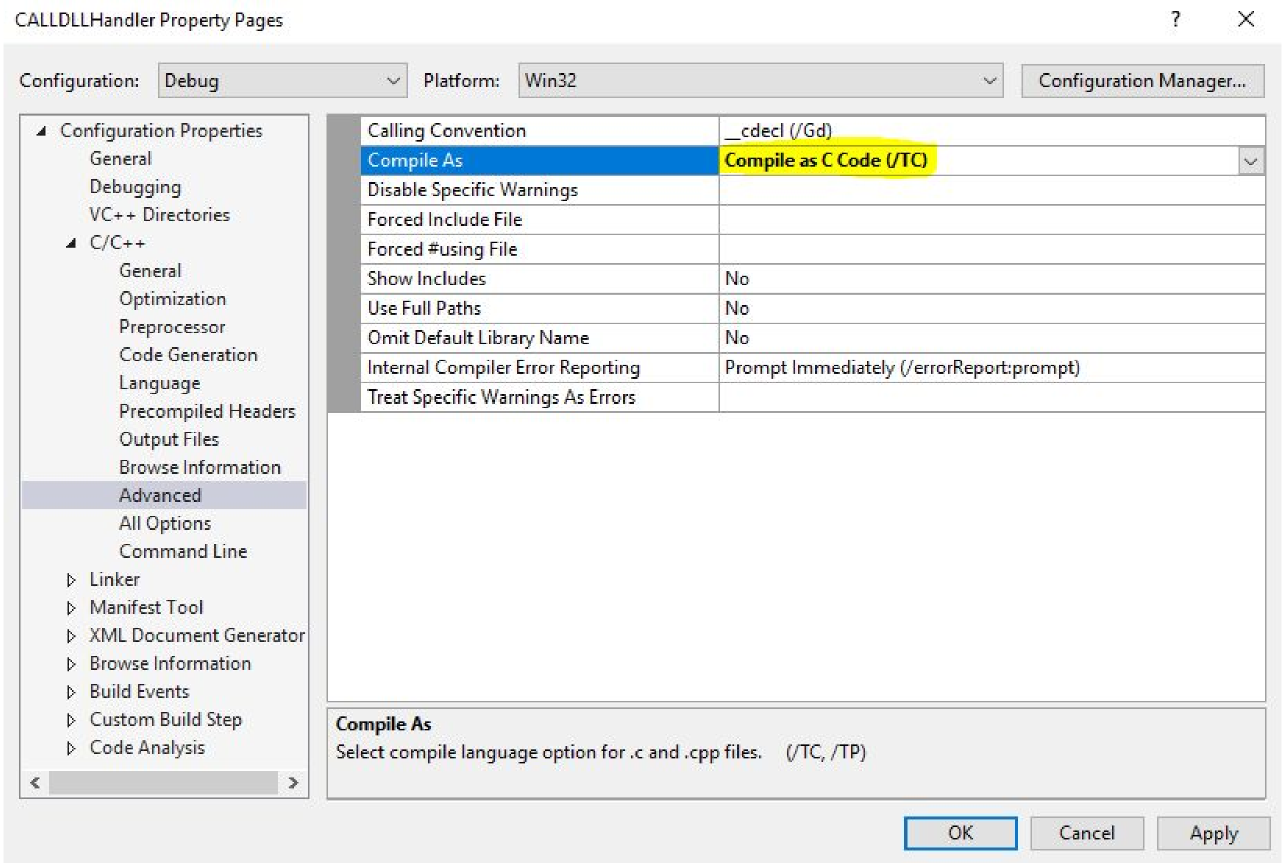


Configuration: Debug
Platform: Win32

Additional Include Directories

```
C:\Users\Public\Documents\Visual Studio Projects\CALLDLLHandler\CALLDLLHandler\include
```

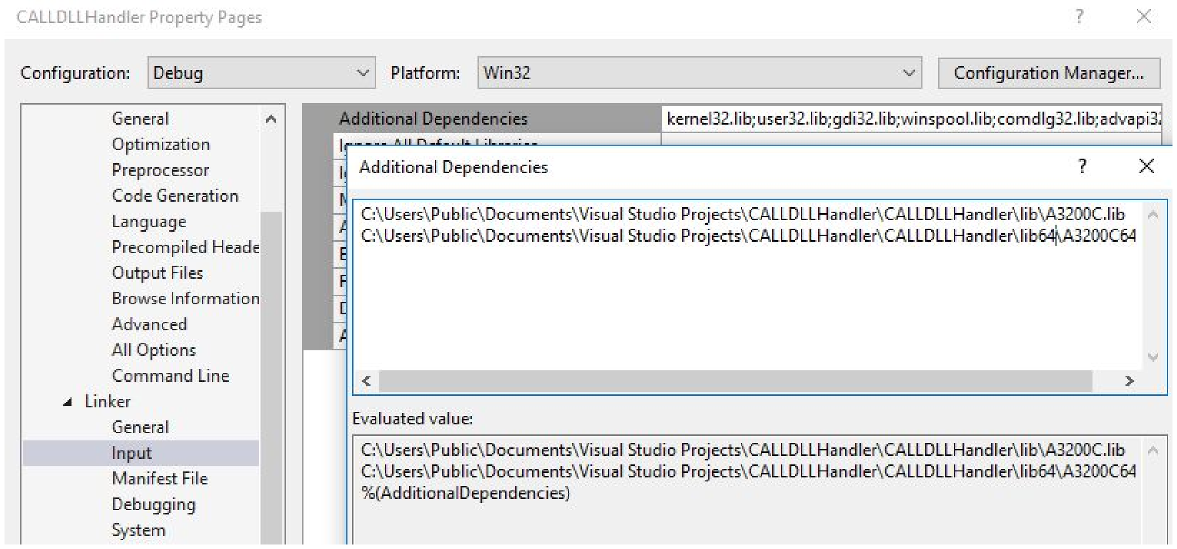
IV. Under “C/C++” -> “Advanced”, change **Compile As** to **Compile as C Code (/TC)**



Calling Convention	__cdecl (/Gd)
Compile As	Compile as C Code (/TC)
Disable Specific Warnings	
Forced Include File	
Forced #using File	
Show Includes	No
Use Full Paths	No
Omit Default Library Name	No
Internal Compiler Error Reporting	Prompt Immediately (/errorReport:prompt)
Treat Specific Warnings As Errors	

Compile As
Select compile language option for .c and .cpp files. (/TC, /TP)

V. Add the paths to both **A3200C.lib** and **A3200C64.lib** to **Linker -> Input -> Additional Dependencies**



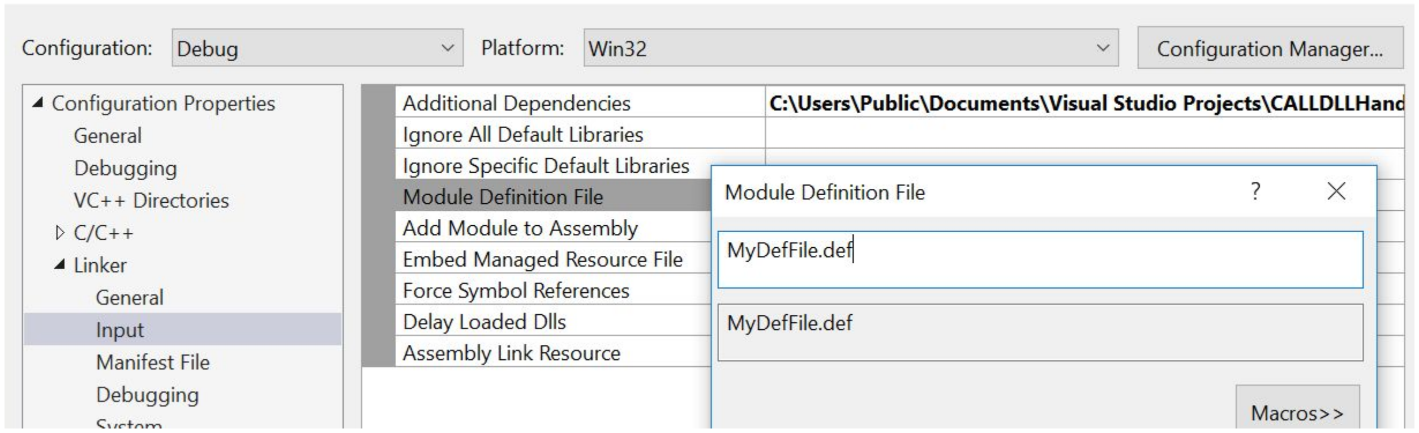
Additional Dependencies: kernel32.lib;user32.lib;gdi32.lib;winspool.lib;comdlg32.lib;advapi3

```
C:\Users\Public\Documents\Visual Studio Projects\CALLDLLHandler\CALLDLLHandler\lib\A3200C.lib
C:\Users\Public\Documents\Visual Studio Projects\CALLDLLHandler\CALLDLLHandler\lib64\A3200C64
```

Evaluated value:

```
C:\Users\Public\Documents\Visual Studio Projects\CALLDLLHandler\CALLDLLHandler\lib\A3200C.lib
C:\Users\Public\Documents\Visual Studio Projects\CALLDLLHandler\CALLDLLHandler\lib64\A3200C64
%(AdditionalDependencies)
```

VI. Add the ***.def file** to **Linker -> Input -> Module Definition File**



Additional Dependencies: **C:\Users\Public\Documents\Visual Studio Projects\CALLDLLHand**

Module Definition File: MyDefFile.def

MyDefFile.def

8. Add includes and function prototypes to the top of the .c Source File

```
#include <stdio.h>

#include <A3200.h>

ErrorData _stdcall FancyEnable(A3200Handle handle, TASKID taskId);
ErrorData _stdcall FancyDisable(A3200Handle handle, TASKID taskId);
```

9. Define the functions and use the **Generic Callback Functions** to handle the passing of data

10. Build the application and address any compiler errors

The function definitions:

```
ErrorData _stdcall FancyEnable(A3200Handle handle, TASKID taskId)
{
	ErrorData errorData = ErrorData_NoError;
	AXISMASK axisMask;
	LPSTR returnedString = "";
	DOUBLE homeAxes;

	if (!A3200CallbackArgsGetInteger(handle, taskId, 2, (INT *)&axisMask))
	{
		errorData = A3200GetLastError();
		goto cleanup;
	}

	if (!A3200CallbackArgsGetDouble(handle, taskId, 3, (DOUBLE *)&homeAxes))
	{
		errorData = A3200GetLastError();
		goto cleanup;
	}
```

```
        if (!A3200MotionEnable(handle, TASKID_Library, axisMask))
        {
                // Enable all of the axes that are specified in the axis mask.
                        errorData = A3200GetLastError();
                goto cleanup;
        }

        //if true, home the axes
        if (homeAxes)
        {
                if (!A3200MotionHome(handle, TASKID_Library, axisMask))
                {
                        // Enable all of the axes that are specified in the axis mask.
                        errorData = A3200GetLastError();
                        goto cleanup;
                }
                returnedString = "Your axes have been enabled and homed";
        }
        else
        {
                returnedString = "Your axes have been enabled";
        }
cleanup:
        A3200CallbackReturnString(handle, taskId, returnedString, errorData, 0, 0);
        return A3200GetLastError();
}

ErrorData _stdcall FancyDisable(A3200Handle handle, TASKID taskId)
{
        ErrorData errorData = ErrorData_NoError;
        AXISMASK axisMask;
        DOUBLE globals[5] = { 0, 0, 0, 0, 0 };
        DOUBLE clearGlobals;

        if (!A3200CallbackArgsGetInteger(handle, taskId, 2, (INT *)&axisMask))
        {
                errorData = A3200GetLastError();
                goto cleanup;
        }
        if (!A3200CallbackArgsGetDouble(handle, taskId, 3, (DOUBLE *)&clearGlobals))
        {
                errorData = A3200GetLastError();
                goto cleanup;
        }
        if (!A3200MotionDisable(handle, TASKID_Library, axisMask))
        {
                // Enable all of the axes that are specified in the axis mask.
                errorData = A3200GetLastError();
                goto cleanup;
        }

        if (clearGlobals)
        {
                A3200VariableSetGlobalDoubles(handle, taskId, globals, 5);
        }

cleanup:
        A3200CallbackReturnVoid(handle, taskId, errorData, 0, 0);
        return A3200GetLastError();
}
```

11. Copy the DLL that is output when building the project and paste it in the same directory as the calling AeroBasic program. *This will allow users to omit the path when using the CALLDLL command and only the *.dll filename is required*

An example of a short AeroBasic program that calls the functions:

```
$strglob[0] = CALLDLL "CALLDLLHandler.dll", "FancyEnable", 3, 1
$global[0] = MSGBOX DF_MSGBOX_OKONLY + DF_ICON_INFORMATION, $strglob[0]
CALLDLL "CALLDLLHandler.dll", "FancyDisable", 3, 1
END PROGRAM
```