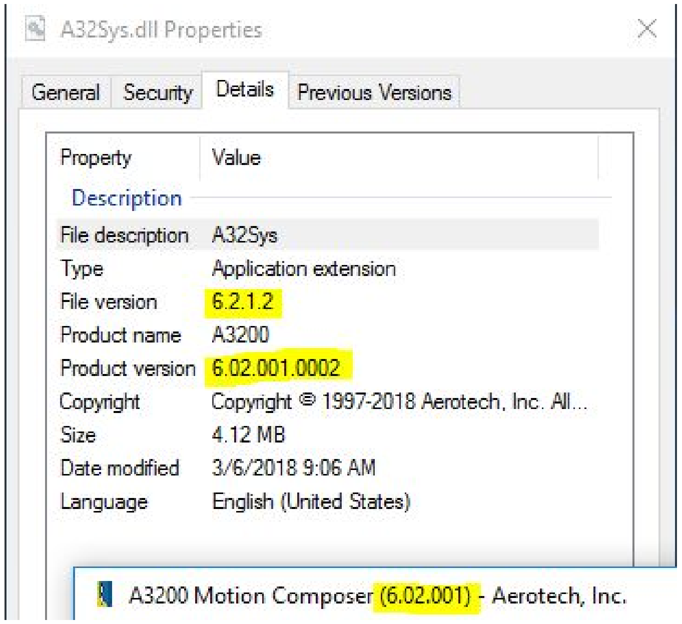
C# Getting Started Guide

This guide will discuss the A3200 .NET Library. The C# language will be used to demonstrate concepts.

Building Projects Against the A3200 .NET Library in Visual Studio:

- The .NET library is built using .NET 2.0 and can be used in applications using .NET 2.0 or newer
- The version of the library DLLs must be in sync with each other and with the version of the A3200 controller installed



Property	Value
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.

- Everything inside of the DotNET directory must be copied into the Output Directory of any .NET application that uses the A3200 .NET Library

- The Aerotech.A3200.dll and Aerotech.Common.dll are managed .NET references that must be added as "Include" to any C# application that uses the A3200 .NET Library (this can be accomplished through the Solution Explorer by right-clicking on References and selecting Add Reference…)

Sample Projects and Code Snippets:

- There are several sample projects located in the Samples directory of the installation directory (e.g. C:\Program Files\Aerotech\A3200\Samples)
- There are example code snippets for the majority of the controller features available through the API
    - Browse the A3200 Programming Help file for classes and methods that wrap A3200 controller features and commands and scroll to the bottom of the page
- The FAQs and example application in this guide are a good starting point

Using the A3200 .NET Library:

- Unless indicated otherwise, the methods in the A3200 .NET 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 .NET 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
    - This includes every method under **Controller.Commands** (this also applies to **Status**)
    - **NewDiagPacketArrived** and **NewTaskStatesArrived** are inherently threaded and any commands executed within these events will need to be synchronized

        - Since these events are raised on a background thread, customers writing GUI applications will need to handle invoking onto the UI thread, if they intend to modify GUI control properties based on incoming status. This process can be accomplished easily with a BackgroundWorker (see https://msdn.microsoft.com/en-us/library/system.componentmodel.backgroundworker.aspx for more information)
            - If a background thread does time-consuming operations, do not directly modify any graphical control properties or call graphical control methods from the background thread (exceptions can occur). Refer to https://docs.microsoft.com/en-us/dotnet/framework/winforms/controls/how-to-make-thread-safe-calls-to-windows-forms-controls to update graphical controls from any thread

- If a Task Error or Axis Fault occurs during a command issued from the A3200 .NET Library, then a .NET exception will be thrown by the method
    - Similarly, if the command cannot be executed because of an existing Task Error or Axis Fault, then a .NET exception will be thrown by the method
    - Axis Faults and Task Errors that occur in an AeroBasic program will not cause .NET exceptions and users are required to monitor their tasks and axes when using the methods in the Program class to run AeroBasic programs from their .NET applications

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 .NET 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 .NET Library, but **users will still need to rebuild their application against the version of the A3200 .NET Library that matches the A3200 controller version** that they intend to use with their application. Follow this procedure when updating .NET applications to function properly with a different controller version:

I. Check the release notes to see what has changed between versions of the A3200 .NET 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’ %A3200InstallDirectory%/DotNET directory and paste them in the application’s Output Directory (the 64-bit versions are needed for 64-bit or AnyCPU applications)

IV. Add the Aerotech.A3200.dll and Aerotech.Common.dll as “Include” to the Visual Studio project

V. Rebuild the application

2. *How do I connect to the controller?*

The **Connect()** method connects to a controller, initializes it (if necessary), and returns the newly connected controller.

```
Controller myController = Controller.Connect();
```

3. *What task will immediate commands issued through the A3200 .NET Library run in?*

By default, immediate commands will execute in the **Library Task**. However, users can specify the task in which they would like their command to execute. This becomes useful when users would like to issue IO commands while a synchronous, blocking, move is being commanded. Motion can be commanded on a different task than IO commands

```
myController.Commands.Axes["Y"].Motion.Enable(); // Enables the Y axis using the Library Task
myController.Commands[TaskId.T01].Motion.Enable(0); // Enables Axis 0 using Task 1
myController.Commands[TaskId.T02].IO.DigitalOutputBit(0, "X", 1); // Sets X axis digital output bit 0 to 1
```

4. *How do I edit controller parameters?*

To edit the parameter values for this instance of the controller, find the parameter that you would like to change in the **Parameters** class and set it through its **Value** property. The controller will revert back to the value in the active parameter file on controller reset.

```
myController.Parameters.Axes["X"].Motion.Home.HomeOffset.Value = 100;
```

5. How do I save a my active parameters to a parameter file?

```
myController.Parameters.SaveToFile(@"C:\Users\Public\Documents\Aerotech\A3200\User Files\ParametersBackup.prma");
```

6. How do I save new parameter values to my active parameter file?

To overwrite the active parameter file, retrieve the path of your active parameter file via the Configuration.ParameterFile property, make desired edits, and then save the current controller parameters to the active parameter file.

```
string paramsPath = Controller.Configuration.ParameterFile;
myController.Parameters.Axes["Y"].Feedback.PositionFeedbackType.Value = 1;
myController.Parameters.SaveToFile(paramsPath);
```

7. How do I read and write controller variables?

See the code snippet below for various examples ($info0, $info1, $return, and other Fieldbus variable types, like Modbus, are available through the library but are not shown).

```
myController.Variables.Global["$global[0]"].Value = 12.34;
myController.Variables.Global.Doubles[1].Value = 43.21;
double[] someGlobalDoubles = myController.Variables.Global.Doubles.GetMultiple(0, 2);
myController.Variables.Global["$strglob[0]"].Value = "hello";
myController.Variables.Tasks[2].Strings[0].Value = "world";
myController.Variables.Tasks[1].Program["$myVariable"].Value = 123;
myController.Variables.Fieldbus.PCModbusMaster[0].OutputWords["$Modbus0Word"].Value = 56;
```

8. How do I handle exceptions?

Try-Catch blocks are a user-friendly way to handle exceptions. Below is an example of try-catch. This example will simply display the error message to the console and unregister from diagnostic packets and callbacks.

```
Controller myController = null;

try
{
     // Connect to A3200.  Initialize if necessary
     myController = Controller.Connect();

     // Register for the NewDiagPacketArrived event, and start data collection (it is started automatically).
     myController.ControlCenter.Diagnostics.NewDiagPacketArrived += new EventHandler<NewDiagPacketArrivedEventArgs>(NewDiagPacketArrived);

     // Try to command motion to the X axis before it is enabled... This will throw an exception
     myController.Commands.Motion.Linear("X", 10);
}
catch(A3200Exception exception)
{
     Console.WriteLine("Error occurred : {0}", exception.Message);
     Console.ReadLine();
}
finally
{
     // Unregister for diagnostic packets
     myController.ControlCenter.Diagnostics.NewDiagPacketArrived -= NewDiagPacketArrived;
}
```

9. *I can’t find the _______ command in the A3200 .NET 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 .NET Library by using the **Execute() method**. 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.

```
var ampTemp = myController.Commands.Execute("DRIVEINFO(X, DRIVEINFO_AmplifierTemperature)");

myController.Commands.Execute("DWELL 1\nWAIT MOVEDONE X\nLINEAR X 10 F 100\nDWELL 1");
```

It is also possible to run AeroBasic .PGMs from a .NET application (e.g. the **Program.Run() method**).

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 **Program.Load() method** to load the program to the controller. The **Execute() method** can be used to call the subroutine from a specific task.

```
myController.Tasks[TaskId.T01].Program.Load(ProgramFileLocation);
myController.Variables.Global["strglob[0]"].Value = "myProgram.pgm";
myController.Variables.Global["strglob[1]"].Value = "myFunction";
myController.Commands.Execute("FARCALL strglob[0] strglob[1]");
```

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 Load() then Start() methods or Run() method, or added to Program Automation in A3200 Configuration Manager as Run or RunSilent.

11. *I can’t find a VELOCITY ON or an equivalent method in the A3200 .NET Library. How do I command a series of moves with velocity profiling in my C# application?*

There is not an API for velocity profiling because the A3200 libraries are not intended for real-time motion control. Users could see between 10-100ms 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 Controller.Commands.Execute(“VELOCITY ON”)*.

Users have two options to command a series of moves with velocity profiling in their .NET application:

I. Write an AeroBasic program that contains a VELOCITY ON command before their sequence of moves and run the program using the **Program.Run() method**
II. Issue Controller.Commands[TaskId.*YourDesiredTask*].Execute(“VELOCITY ON”) to enable velocity blending and then place this task in **Queue Mode**. In Queue Mode, motion commands will be executed as quickly as possible, asynchronously (Motion.Linear will no longer block), and the user will be responsible for synchronizing their application the the controller’s queue status (see the “Queue Mode Overview” topic in the A3200 Help file for more information)

12. *Can the A3200 .NET Library be used to control a Hexapod?*

With the Hexapod files setup correctly with Program Automation, the **Execute() method** can be used to issue Hexapod commands to do things like setup and activate tools and 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 **Program.Run() method** can be used to run AeroBasic programs that contain the VELOCITY ON command followed by the sequence of moves (See FAQ #9).

13. How do I poll for diagnostics and status items through the A3200 .NET Library?

For a brief overview, see the ".NET Status Retrieval" topic in the A3200 Programming Help file. In general, users have five options to retrieve status items:

I. The methods in the **StatusCommands class** can be used to retrieve a specific item at a single point in time. The methods in this class are technically commands so they need to be synchronized.

```
double poscmd = myController.Commands.Status.AxisStatus("X", AxisStatusSignal.PositionCommand);
double inposition = myController.Commands.Status.AxisStatus("X", AxisStatusSignal.DriveStatus, 0x02000000);
double mfo = myController.Commands.Status.TaskStatus(TaskId.T01, TaskStatusSignal.MFO);
double virtinpt0 = myController.Commands.Status.SystemStatus(SystemStatusSignal.VirtualBinaryInput, 0);
```

II. Subscribe to the **NewDiagPacketArrived** and/or **NewTaskStatesArrived** events. These events are raised periodically (the default is 100 ms) on a separate thread. See the **ControllerDiagPacket class** to see the items within the packets. Data accessed and commands issued within the event handlers must be synchronized. This approach is good for updating a GUI with diagnostic information.

```
public static void Main()
{
    Controller myController = null;

    try
    {
        // Connect to A3200.  Initialize if necessary
        myController = Controller.Connect();

        // Register for the NewDiagPacketArrived event, and start data collection (it is started automatically).
        myController.ControlCenter.Diagnostics.NewDiagPacketArrived += new EventHandler<NewDiagPacketArrivedEventArgs>(NewDiagPacketArrived);
    }
    catch(A3200Exception exception)
    {
        Console.WriteLine("Error occurred : {0}", exception.Message);
        Console.ReadLine();
    }
    finally
    {
        // Unregister for diagnostic packets
        myController.ControlCenter.Diagnostics.NewDiagPacketArrived -= NewDiagPacketArrived;
    }
}
public static void NewDiagPacketArrived(object sender, NewDiagPacketArrivedEventArgs diagPacket)
{
    // For each axis, check if faults exist, and then check for specific and generic faults.
    foreach (AxisDiagPacket axisDiagPacket in diagPacket.Data)
    {
        if (!axisDiagPacket.AxisFault.None)
        {
            // To check for a specific fault, you can do this:
            if (axisDiagPacket.AxisFault.PositionErrorFault)
            {
                // Display the position error fault and the axis.
                Console.WriteLine("!! Position Error Fault on Axis: {0}", axisDiagPacket.AxisName);
            }
            // To show all faults, do this:
            Console.WriteLine("Current Axis Faults for Axis {0}: {1}", axisDiagPacket.AxisName, axisDiagPacket.AxisFault);
            // From here, you can choose to clear the fault or take some action based on the fault.
        }
    }
}
```

III. When time critical diagnostics are required, users can build groups of multiple status items with **CustomDiagnostics** and use the **Retrieve() method** to retrieve these items. The Retrieve() method takes a snapshot of all of the diagnostic items within the CustomDiagnostics packet at that point in time. Users can access individual items and their values at the point in time in which the packet was retrieved. This is a thread safe way to poll diagnostic items.

```
// Configure the custom diagnostics
CustomDiagnostics customDiagnostics = new CustomDiagnostics(myController);
customDiagnostics.Configuration.Axis.Add(AxisStatusSignal.DriveStatus, "X");
customDiagnostics.Configuration.Axis.Add(AxisStatusSignal.DriveStatus, "Y");
customDiagnostics.Configuration.Axis.Add(AxisStatusSignal.PositionCommand, "X");
customDiagnostics.Configuration.Axis.Add(AxisStatusSignal.PositionCommand, "Y");
customDiagnostics.Configuration.Variable.Add("$global[0]");
customDiagnostics.Configuration.System.Add(SystemStatusSignal.GlobalVariable, 5);
// Get the values
CustomDiagnosticsResults result = customDiagnostics.Retrieve();
// Extract the values, note that they can be extracted in any order
Console.WriteLine("Position Command for Axis X: {0}", result.Axis[AxisStatusSignal.PositionCommand, "X"].Value);
Console.WriteLine("Position Command for Axis Y: {0}", result.Axis[AxisStatusSignal.PositionCommand, "Y"].Value);
Console.WriteLine("Drive Status for Axis X: {0}", result.Axis[AxisStatusSignal.DriveStatus, "X"].ConvertValueDriveStatus());
Console.WriteLine("Drive Status for Axis Y: {0}", result.Axis[AxisStatusSignal.DriveStatus, "Y"].ConvertValueDriveStatus());
Console.WriteLine("Global Double #0 : {0}", result.Variable["$global[0]"].Value);
Console.WriteLine("Global Double #5 : {0}", result.System[SystemStatusSignal.GlobalVariable, 5].Value);
```

IV. If the user desires a specific set of diagnostic items over a certain period of time, they can configure a **Data Collection**. GetData() blocks until the Data Collection is complete.

```
DataCollectionConfiguration config = myController.DataCollection.Configuration;
config.Axis.Add(AxisDataSignal.PositionCommand, "X");
config.Axis.Add(AxisDataSignal.PositionFeedback, "X");
config.AxisExtended.Add(AxisExtendedDataSignal.AmplifierTemperature, "X");
// Collect 1 point of data for the signals every 2 msec
myController.DataCollection.Configuration.SampleTrigger.Time = new SampleTriggerTimeConfiguration(2.0, TimeUnit.Milliseconds);
// Collect 1,000 points of data for each signal
config.PointsToCollect = 1000;
// Start the data collection
myController.DataCollection.Start();
```

```
// Retrieve the data points collected (blocks until finished)
DataCollectionResults results = myController.DataCollection.GetData();
// Print out the 75th position command point collected
Console.WriteLine("Amplifier Temperature of Axis 0: {0}", results.AxisExtended[AxisExtendedDataSignal.AmplifierTemperature, "X"].Points[0x4a]);
```

V. The **Execute() method** can be used to issue AeroBasic status commands as immediate commands. The Execute() method is not thread safe and users must use locks or some other means of synchronization if they are using threading.

```
var inposition = myController.Commands.Execute("AXISSTATUS(X, DATAITEM_DriveStatus, DRIVESTATUS_InPosition)");
```

14. How do I control and monitor the execution of existing AeroBasic .PGM or .OGM files from my C# application?

Through the **Program** and **LoadedProgram** classes, users can run, load/unload, associate, start, stop, and pause AeroBasic programs from within their .NET application. The Run() and Start() methods do not block so the user will be responsible for monitoring the state of the task to know when the program has finished executing. Axis Faults and Task Errors that occur while an AeroBasic program is running will not cause .NET exceptions and the user should monitor the task and axes for errors.

```
using System;
using Aerotech.A3200;
using Aerotech.A3200.Status;
using Aerotech.A3200.Exceptions;

public static class Program
{

    public static string ProgramFileLocation = @"C:\Users\Public\Documents\Aerotech\A3200\User Files\Test.pgm";

    public static void Main()
    {
        Controller myController = null;

        try
        {
            // Connect to A3200.  Initialize if necessary
            myController = Controller.Connect();
            // Register for the NewDiagPacketArrived event, and start data collection (it is started automatically).
            myController.ControlCenter.Diagnostics.NewDiagPacketArrived += new EventHandler<NewDiagPacketArrivedEventArgs>(NewDiagPacketArrived);
            // Register for the NewTaskStatesArrived event, and start data collection (it is started automatically).
            myController.ControlCenter.TaskStates.NewTaskStatesArrived += new EventHandler<NewTaskStatesArrivedEventArgs>(NewTaskStatesArrived);
            // Register for the ErrorOccurred event to process any occurs that may occur during polling.
            myController.ControlCenter.Diagnostics.ErrorOccurred += new EventHandler<ErrorEventArgs>(NewErrorOccurred);
            // Run an AeroBasic program in Task 1. The Run() method is non-blocking.
            myController.Tasks[TaskId.T01].Program.Run(Program.ProgramFileLocation);

            Console.ReadLine();
        }
        catch (A3200Exception exception)
        {
            Console.WriteLine("Error occurred : {0}", exception.Message);
            Console.ReadLine();
        }
        finally
        {
            //unsubscriptions
            myController.ControlCenter.Diagnostics.NewDiagPacketArrived -= NewDiagPacketArrived;
            myController.ControlCenter.TaskStates.NewTaskStatesArrived -= NewTaskStatesArrived;
            myController.ControlCenter.Diagnostics.ErrorOccurred -= NewErrorOccurred;
        }
    }
    public static void NewDiagPacketArrived(object sender, NewDiagPacketArrivedEventArgs diagPacket)
    {
        // For each axis, check if faults exist, and then check for specific and generic faults.
        foreach (AxisDiagPacket axisDiagPacket in diagPacket.Data)
        {
            if (!axisDiagPacket.AxisFault.None)
            {
                // To check for a specific fault, you can do this:
                if (axisDiagPacket.AxisFault.PositionErrorFault)
                {
                    // Display the position error fault and the axis.
                    Console.WriteLine("!! Position Error Fault on Axis: {0}", axisDiagPacket.AxisName);
                }
                // To show all faults, do this:
                Console.WriteLine("Current Axis Faults for Axis {0}: {1}", axisDiagPacket.AxisName, axisDiagPacket.AxisFault);
                // From here, you can choose to clear the fault or take some action based on the fault.
            }
        }
    }

    public static void NewErrorOccurred(object sender, ErrorEventArgs errorEventArgs)
    {
        // Print the error that occurred while polling data.
        Console.WriteLine("Error occurred while polling data: {0}", errorEventArgs.Exception.Message);
    }

    public static void NewTaskStatesArrived(object sender, NewTaskStatesArrivedEventArgs taskStates)
    {
        // Print out the current task state for task 1
        Console.WriteLine("Task 1's State: {0}", taskStates.TaskStates[TaskId.T01]);
    }
}
```

15. Which A3200 .NET Library methods are blocking?

Unless indicated otherwise, the A3200 .NET Library methods have the same blocking behavior as their AeroBasic equivalents. If the method in question is not a wrapper for an AeroBasic command, the blocking behavior will be specified in the method's help file topic. Please see the "**Blocking Behavior of the Libraries**" topic in the A3200 Programming Help file for more information.

16. Is it possible to get around the blocking behavior of methods in the A3200 .NET Library and allow program execution to continue through a method that has blocking behavior?

Users can place a controller task in **Queue Mode** and add the command in question to the task's execution queue (see the "Queue Mode Overview" topic in the A3200 Help file for more information). Another option is to write an AeroBasic program that contains the desired commands and use the methods in the **Program class** to load, start, and stop the program. The Run() method does not block.

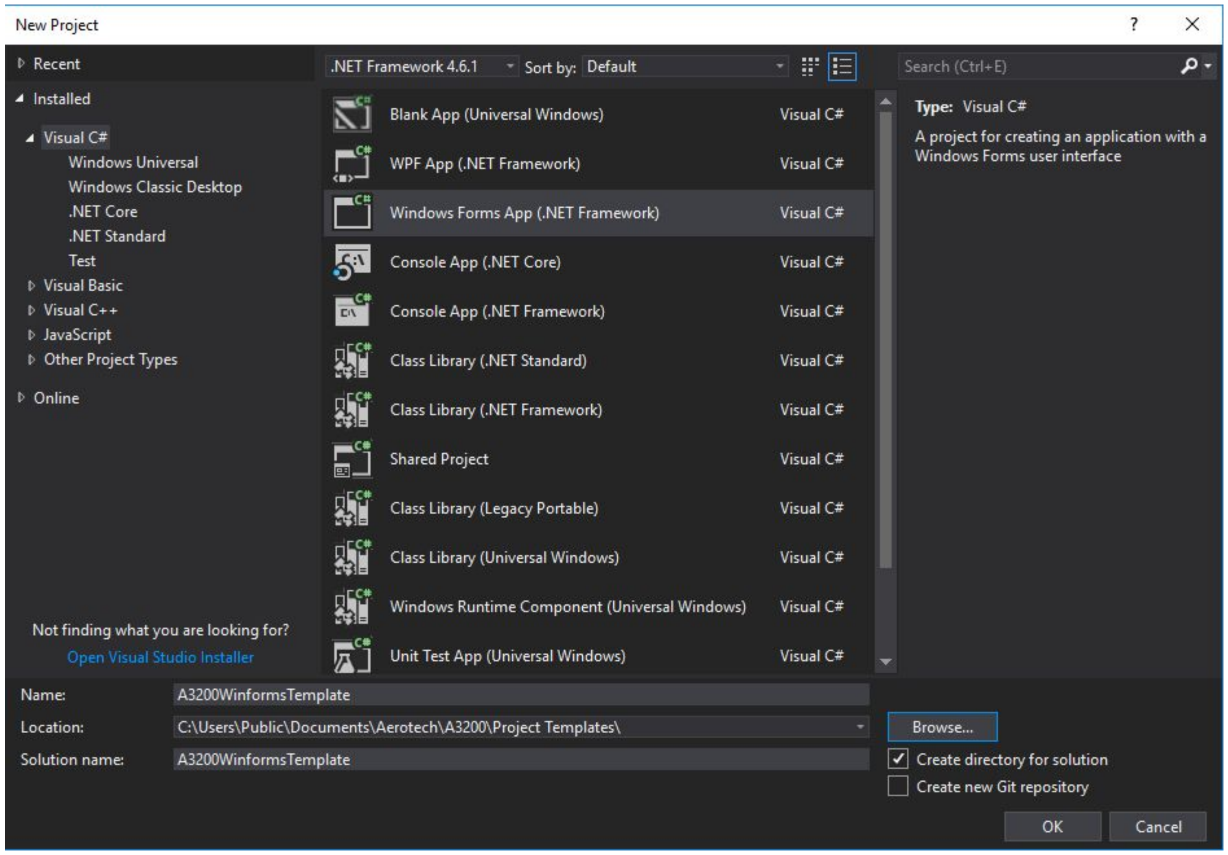
If the commands in question are not available as commands in A3200 AeroBasic or if the user would like to solve this problem using features of .NET, they can instantiate a .NET Thread object and define a function or delegate that contains the blocking method. This is useful when the user would like to allow their UI thread to perform another task while the blocking method executes (see the "**Blocking Behavior of the GUIs"** topic in the A3200 Programming Help file for more information). However the user is responsible for making their application thread safe (by introducing locks or some using other method to synchronize) and commands cannot be executed from two threads at the same time.

Generating a Project Template in Visual Studio:

1. Open Visual Studio

2. File -> New Project

3. Select a project type, name, and location (Winforms was chosen for this example)



New Project

- Recent
- Installed
  - Visual C#
    - Windows Universal
    - Windows Classic Desktop
    - .NET Core
    - .NET Standard
    - Test
  - Visual Basic
  - Visual C++
  - JavaScript
  - Other Project Types
- Online

Not finding what you are looking for?
Open Visual Studio Installer

.NET Framework 4.6.1 Sort by: Default

Search (Ctrl+E)

Blank App (Universal Windows)	Visual C#
WPF App (.NET Framework)	Visual C#
Windows Forms App (.NET Framework)	Visual C#
Console App (.NET Core)	Visual C#
Console App (.NET Framework)	Visual C#
Class Library (.NET Standard)	Visual C#
Class Library (.NET Framework)	Visual C#
Shared Project	Visual C#
Class Library (Legacy Portable)	Visual C#
Class Library (Universal Windows)	Visual C#
Windows Runtime Component (Universal Windows)	Visual C#
Unit Test App (Universal Windows)	Visual C#

Type: Visual C#

A project for creating an application with a Windows Forms user interface

Name: A3200WinformsTemplate

Location: C:\Users\Public\Documents\Aerotech\A3200\Project Templates\

Solution name: A3200WinformsTemplate

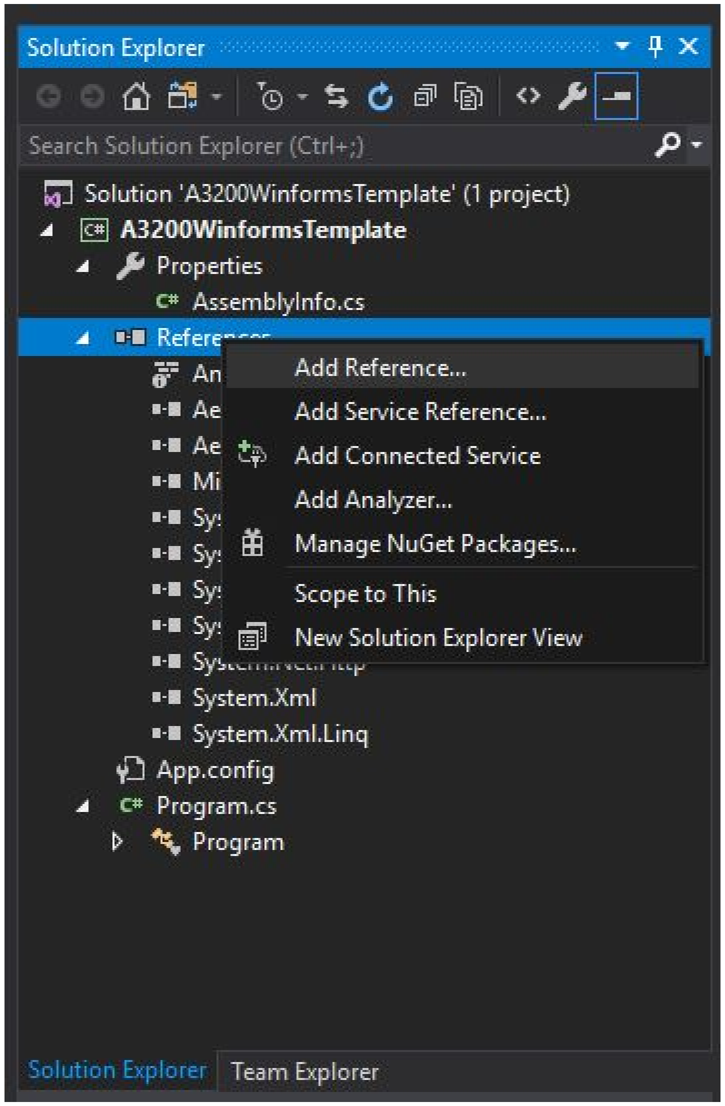
Browse...

[✓] Create directory for solution

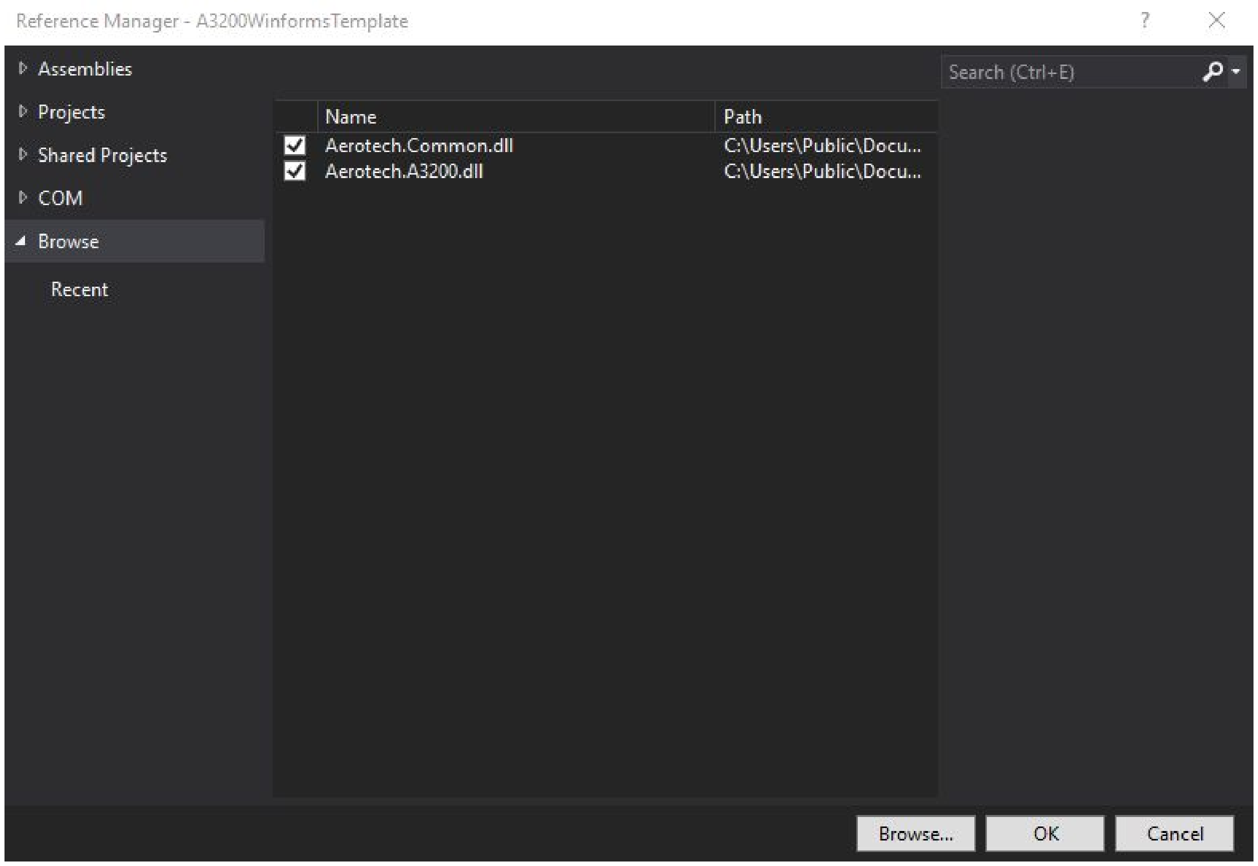
[ ] Create new Git repository

OK Cancel

4. Right click on References in the Solution Explorer and select Add Reference…

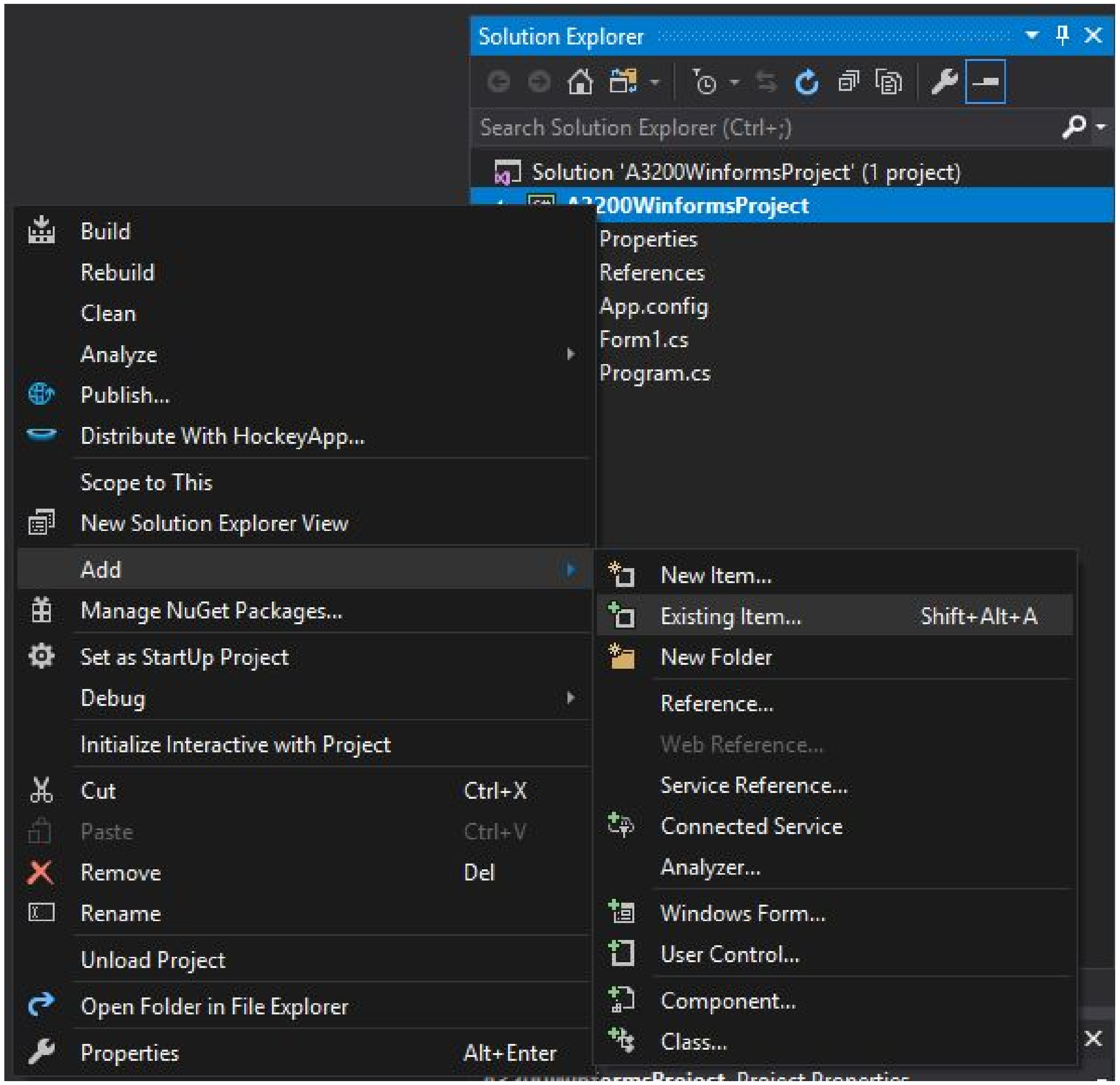


5. Click Browse… and navigate to A3200InstallDirectory\release\DotNET and add Aerotech.A3200.dll and Aerotech.Common.dll as references

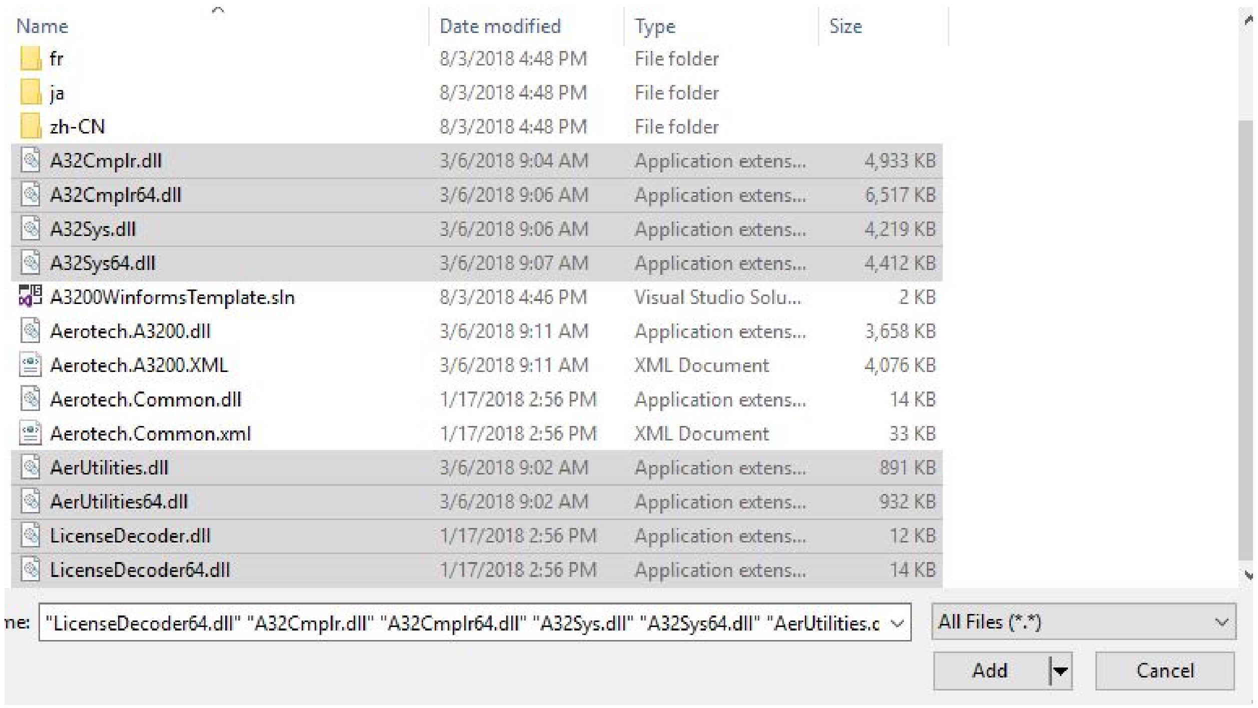


	Name	Path
✓	Aerotech.Common.dll	C:\Users\Public\Docu...
✓	Aerotech.A3200.dll	C:\Users\Public\Docu...

6. Right click on the project name in the Solution Explorer and select Add -> Existing Item



7. Select both versions of the A32Cmply, A32Sys, AerUtilities, and LicenseDecoder DLLs and add them to the Solution. Select multiple files by pressing the Ctrl key while selecting them. Adding both versions will make it possible to target Any CPU

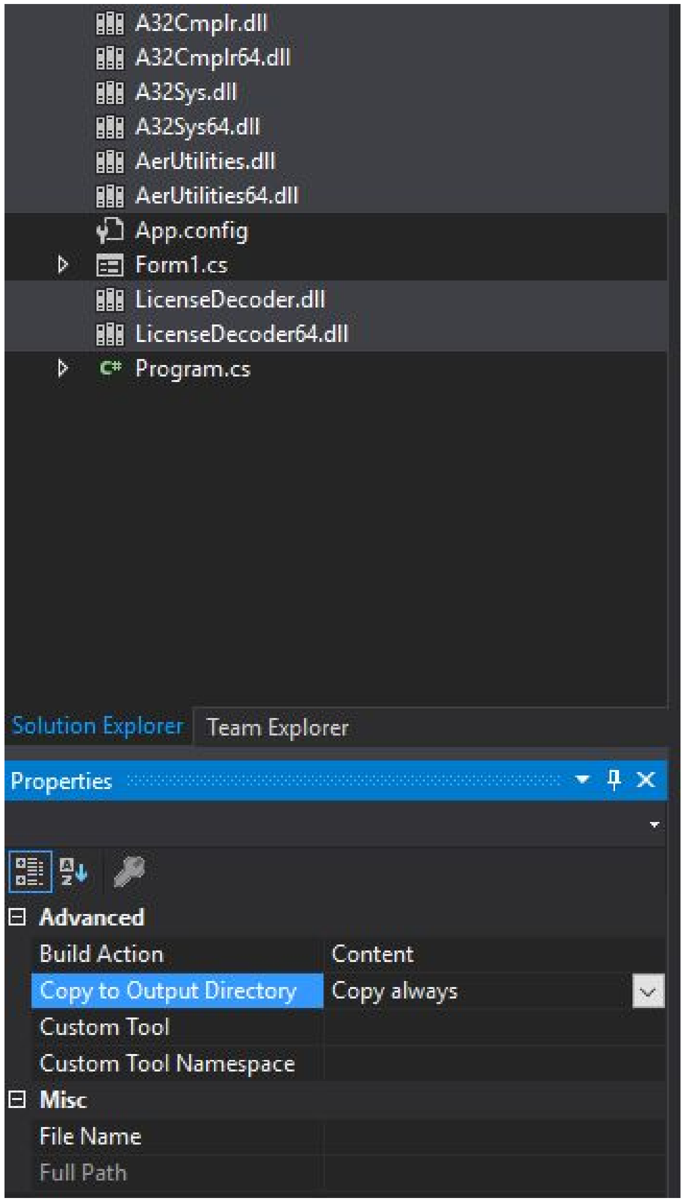


Name	Date modified	Type	Size
fr	8/3/2018 4:48 PM	File folder	
ja	8/3/2018 4:48 PM	File folder	
zh-CN	8/3/2018 4:48 PM	File folder	
A32Cmplr.dll	3/6/2018 9:04 AM	Application extens...	4,933 KB
A32Cmplr64.dll	3/6/2018 9:06 AM	Application extens...	6,517 KB
A32Sys.dll	3/6/2018 9:06 AM	Application extens...	4,219 KB
A32Sys64.dll	3/6/2018 9:07 AM	Application extens...	4,412 KB
A3200WinformsTemplate.sln	8/3/2018 4:46 PM	Visual Studio Solu...	2 KB
Aerotech.A3200.dll	3/6/2018 9:11 AM	Application extens...	3,658 KB
Aerotech.A3200.XML	3/6/2018 9:11 AM	XML Document	4,076 KB
Aerotech.Common.dll	1/17/2018 2:56 PM	Application extens...	14 KB
Aerotech.Common.xml	1/17/2018 2:56 PM	XML Document	33 KB
AerUtilities.dll	3/6/2018 9:02 AM	Application extens...	891 KB
AerUtilities64.dll	3/6/2018 9:02 AM	Application extens...	932 KB
LicenseDecoder.dll	1/17/2018 2:56 PM	Application extens...	12 KB
LicenseDecoder64.dll	1/17/2018 2:56 PM	Application extens...	14 KB

ne: "LicenseDecoder64.dll" "A32Cmplr.dll" "A32Cmplr64.dll" "A32Sys.dll" "A32Sys64.dll" "AerUtilities.c

All Files (*.*)

8. Select all of the DLLs and set the Copy to Output Directory Property to “Copy always”

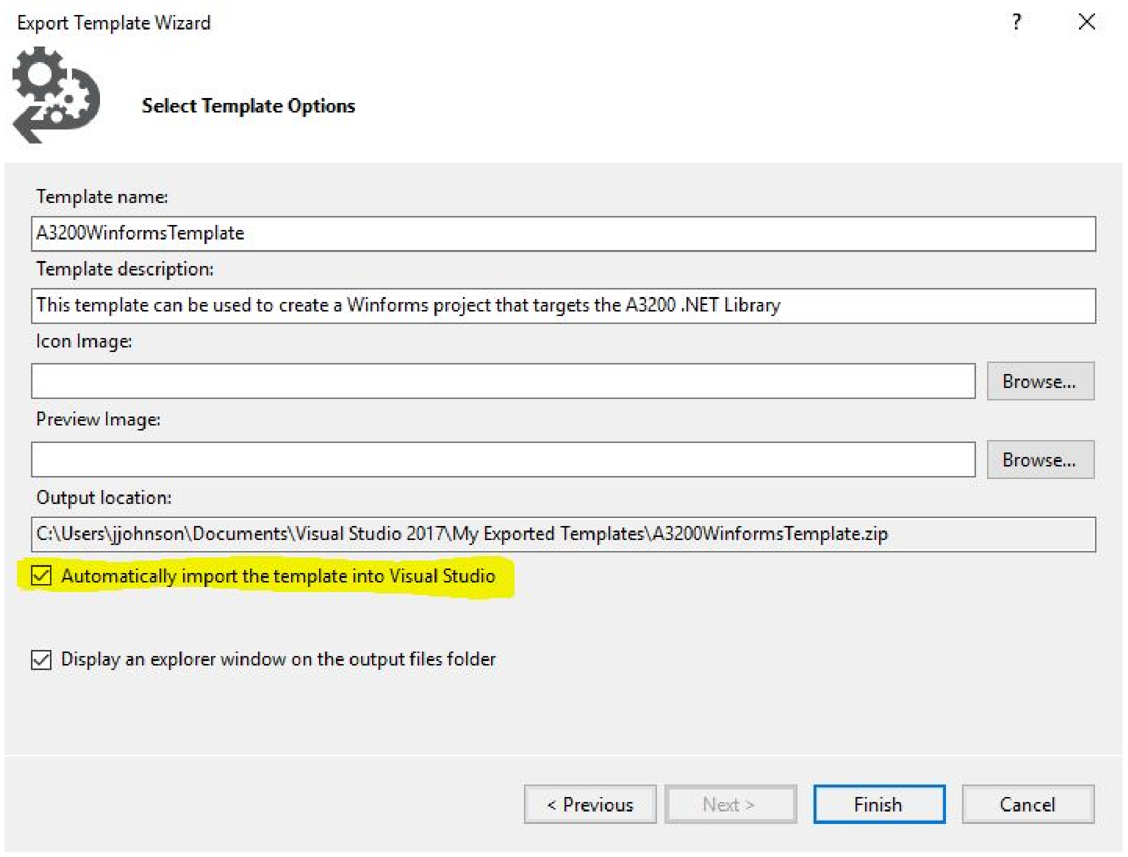


Build Action	Content
Copy to Output Directory	Copy always
Custom Tool	
Custom Tool Namespace	
File Name	
Full Path	

9. Save your project and navigate to Project -> Export Template to open the Export Template Wizard

10. Make sure that Project template is selected and click Next >

11. Customize template name, description, icon, etc. and check the box next to "Automatically import the template into Visual Studio" before clicking Finish



Template name:
A3200WinformsTemplate
Template description:
This template can be used to create a Winforms project that targets the A3200 .NET Library
Icon Image:
Preview Image:
Output location:
C:\Users\jjohnson\Documents\Visual Studio 2017\My Exported Templates\A3200WinformsTemplate.zip
☑ Automatically import the template into Visual Studio
☑ Display an explorer window on the output files folder

12. Once a template has been created, it can be selected when creating a new project