# About

Adhesion Test is a WPF (Windows Presentation Foundation) application developed in C# to allow for the automated testing of adhesives, using a series of force probes and actuators. This software was initially designed to function with a normal and a shear force probe each connected to a transducer and then to a national instruments USB hub (NI USB-6289), the actuators used were the MFA-CC model produced by Newport, which are controlled by an ESP-301 motor controller hub. Three actuators were used allowing for motion in all three axes. This application uses Oxyplot as a graphing library and mahapps.metro as a UI toolkit for WPF. Links to all of these items are given in the resources section of this documentation.

This application uses .NET assemblies distributed by National Instruments to communicate with the national Instruments hub, but due to the lack of availability of similar resources for the ESP 301 hub, implements a custom USB interface for communication with the motor controller.

This software was initially developed over the summer of 2016 by Samuel Lehmann. If you wish to contact him, you may try to do so at [sjlehman@ualberta.ca](mailto:sjlehman@ualberta.ca).

**Also, please note that a complete documentation of all functions, classes, variables and so on in this project is included in a help.chm file, located in the help folder of the project.**

# How to develop on this application

This software is developed in C#, an object oriented .NET language highly reminiscent of Java. Although this document attempts to clearly outline the processes necessary to communicate with the peripheral devices effectively, in order to develop further software one should have some programming knowledge, and an understanding of object-oriented programming (OOP). This is a WPF application, meaning that XAML is used to design the layouts of the application. XAML is a markup language highly similar to XML, and isn’t complicated to understand. In order to develop this application, Visual Studio must be used, with all of the required features to develop for C#. Visual Studio comes with a drag and drop editor that will somewhat aid design of UI through XAML.

# Prior to use of the program

Ensure that all peripherals are connected correctly, and that their drivers have been correctly installed. Furthermore, in the settings of the program (accessed on the top right of the window) ensure that the communication port for the motor controller is correct, and also that the channels for each force probe is correct.

# Known Issues

When stopping a trial early, the probe will occasionally continue its descent into the sample.

# Communication with the NI hub

In order to communicate with the NI hub, additional assemblies were downloaded, (links available in the resources section), and references were added. The two references needed to communicate with the hub are: NationalInstruments.Common and NationalInstruments.DAQmx. In order to access these assemblies, one must download the NI DAQmx driver (see resources), being certain to select .NET support during install. This requires a custom installation. An extensive tutorial as to how to acquire data from the NI hub is linked in the resources section of this document.

### How to use:

First one must create a DAQ (data acquirer) object:



Where the normalChannelId and shearChannelId objects are strings representing the channel that you wish to use, for example: “dev1/ai0”. A wiring diagram can be found within the National Instruments documentation, or their software.

Following this a class named Constants must be created or modified to contain the following values, using 1 and 5 in this example:



After constructing the DAQ object, data will start collecting automatically. Data is saved into a list named dataPoints of structs of type dataPoint. The dataPoint struct is defined as so:



Where the time characteristic is the number of seconds since the creation of the object. As an example, in order to access the most recent dataPoint, one could do the following:



Both the normal and shear data outputted by the dataAcquirer is modified by two factors, the offset and the gain. Offset is simply the vertical displacement of the data, done by simple addition, and gain is the vertical stretch of the data, done by multiplication. It is important to note that a dataPoint is multiplied by the gain before it is shifted by the offset. In order to set the gain and normal values one can do the following:



Offset values are set to zero by default, whereas gain values are set to 98.1 by default.

Additionally, the DAQ class produces a dataAcquired event when a datapoint is collected, which can be subscribed to like so:



Where the function that you wish to run looks like so:



### Variables

|  |  |
| --- | --- |
| **Normal Gain** | A multiplicative factor that applies to the normal data. Is implemented before the offset. |
| **Normal Offset** | An additive factor that applies to the normal data. Is implemented after the gain. |
| **Shear Gain** | A multiplicative factor that applies to the shear data. Is implemented before the offset. |
| **Shear Offset** | An additive factor that applies to the shear data. Is implemented after the gain. |
| **Normal Channel Name** | Is set in the Constants file, indicating the name to be given to the normal channel. |
| **Shear Channel Name** | Is set in the Constants file, indicating the name to be given to the shear channel. |
| **Number of Samples** | The number of samples to be linearly averaged per data point collected. Using a value of 1 will result in the most data points, but greater noise. Is set in the Constants class. |
| **DAQFREQ** | The delay in milliseconds between the collection of samples. Is set in the Constants class. |

# Communication with the ESP 301 Motor Controller

As the motor controller does not have a set of assemblies that are available like the National Instruments hub, a USB interface was designed using C#’s serial namespace. The ESP-301 device can send and receive a variety of commands. In order to send or receive data from the device a 2 letter code is sent to the device. These codes are discussed inn the ESP-301 documentation, linked in the resources section of this document. The majority of the functions that the motor controller is capable of are implemented in the espManager class.

### How to Use

First one must generate the espManager object



Then the espManager is all setup and ready to use. It is generally a good idea to turn on the motors before their use, so as an example:



Would turn the motors on for all of the axes.

# Resources

[Link to Mahapps](http://mahapps.com/)

[Link to Oxyplot](http://www.oxyplot.org/)

[Link to MFA-CC](https://www.newport.com/f/miniature-mfa-motorized-linear-stages)

[Link to ESP-301](https://www.newport.com/f/esp301-3-axis-dc-and-stepper-motion-controller)

[ESP-301 Documentation](http://assets.newport.com/webDocuments-EN/images/ESP301_User_manual.PDF)

[NI DAQmx download - may be antiquated](http://www.ni.com/download/ni-daqmx-15.1.1/5665/en/)

[NI DAQ tutorial](http://home.hit.no/~hansha/documents/microsoft.net/tutorials/data%20acquisition%20in%20csharp/Data%20Acquisition%20in%20CSharp.pdf)