Installation notes for EDM Suite

Jimmy Stammers and Jack Devlin

13/4/2016

1 General Information

The EDMSuite was originally written to control a molecular beam experiment and over the years has been developed into a more general-purpose control software. The bulk of the code is designed to be independent of hardware, to remove the dependency on build requirements for specific computers. The references to hardware are included in higher-levels of the code.

2 Installation Instructions

2.1 Requirements

- Microsoft Visual Studio 2013, install from
  https://imperial.onthehub.com/WebStore/ProductsByMajorVersionList.aspx, sign in with imperial ID
- National Instruments Measurement Studio 2015, download from NI. License code is in 015, Bay 2 filing cabinet.
- National Instruments drivers, these depend on the hardware you want to run. Look at the pages on the NI website for the cards that you have. These will definitely include NI-DAQmx. Use the latest version of these drivers. Tested successfully with DAQmx version 15.5.45.109
- git, download from https://git-scm.com/
- edmSuite - available from http://github.com/coldmatter/EDMSuite

2.2 Steps for Installation

1. Install Visual Studio 2013
2. Install Measurement Studio 2015
3. Launch Visual Studio and the edmSuite solution `EDMSuite.sln`. When first opening, a dialogue may ask if you would like to automatically update references to newer installed versions.
4. Every project requires a `licenses.licx` file, which is generated by measurement studio. This can be done automatically using the generate licenses command in the Measurement Studio tab, but you may need to manually create a blank file in each project folder.
5. Choose the configuration `analysis` which does not depend on any hardware. Build this to verify correct installation.

3 Configuring EDMSuite for the Specific Computer

3.1 Defining the Computer as an Environment

In the `DAQ.environs` method add a `case` of the following form

```
case "your-computer":
        Hardware = new YourHardware();
        FileSystem = new YourFileSystem();
        Debug = false;
        break;
```

3.2 Defining the Hardware Class

1. Create a new class in the DAQ project `YourHardware.cs`. This class requires `NationalInstruments.DAQmx` and `DAQ.pattern` and is in `namespace DAQ.HAL`. The class inherits from `DAQ.HAL.Hardware`. DAQ boards are added using

   ```
   Boards.Add("namestring","boardlocation");
   ```

   where `"boardlocation"` is the device name in NI-MAX.

2. Define a string for each board, e.g.

   ```
   string boardName = (string)Boards["namestring"];
   ```

3. Add Digital/Analogue I/O channels, e.g.

   ```
   AddDigitalOutputChannel("name", boardName, portNo, lineNo);
   ```

4. You can look at `PXIEDMHardware.cs` for an example

3.3 Defining the File System

1. Create a new class in the DAQ project `YourFileSystem.cs`. This class is in `namespace DAQ` and inherits from `DAQ.Environment.FileSystem`.

2. Paths are added in the following way

   ```
   Paths.Add("yourPath","drive:\\folder\\subfolder")
   ```

3. Add data paths to the `DataSearchPaths` object

   ```
   DataSearchPaths.Add(Paths["yourDataPath"]);
   ```

3.4 Make a new Configuration

1. Navigate to the configuration manager
2. Add a new build configuration and select the required projects. This must at least have `DAQ`, `SharedCode`.
3. Now build this configuration to check that you have all the NI drivers. If there are references that are not found, you may need to download additional drivers.

4 Controlling your Hardware

Create a class in DAQ called `YourHardwareController.cs`. Set this to be the start-up project.

4.1 Creating and Modifying Tasks

- Each channel must be defined as an object of the type `Task`, which is located in `NationalInstruments.DAQmx`. These tasks are created using methods of the following form

```
private Task CreateAnalogInputTask(string channel)
        {
            Task task = new Task("NAVHCIn"+channel);
            ((AnalogInputChannel)Environs.Hardware.AnalogInputChannels[channel]).AddToTas
                task,
                0,
                10
            );
            task.Control(TaskAction.Verify);
            return task;
        }
```

  With similar methods for digital and/or output tasks. It may also be useful to define methods which create analogue tasks with a specified range. In the above case, the low value is set to 0 and the high is set to 10.

- Set methods are defined in a similar way

```
private void SetAnalogOutput(Task task, double voltage)
        {
            AnalogSingleChannelWriter writer = new
               AnalogSingleChannelWriter(task.Stream);
            writer.WriteSingleSample(true, voltage);
            task.Control(TaskAction.Unreserve);
        }
```

  as are read methods

```
private double ReadAnalogInput(Task task)
        {
            AnalogSingleChannelReader reader = new
               AnalogSingleChannelReader(task.Stream);
            double val = reader.ReadSingleSample();
            task.Control(TaskAction.Unreserve);
            return val;
        }
```

  For analogue channels, there is also a method which reads multiple samples at a user-defined sampling rate and returns their mean value.

- When the digital tasks are created, they are added to Hash tables `digitalTasks` or `digitalInputTasks`, so these must also be defined

4.2 Configuring the GUI

In `YourHardwareController`, create a method `public void Start()`, which is the initialises the hardware controller. All the analogue and digital tasks are created here and the window is created using `ControlWindow()`. After this, define methods `internal void WindowLoaded()` and `internal void WindowClosed()` which run after starting and closing the application respectively. In the controller window, a toolbox can be used to add components such as buttons, text fields and check boxes. As an example, I created a text box named `bFieldTextBox` and defined a method in the controller class `BFieldCurrent` which parses the value from this text box. A function `UpdateBfield()` then outputs this as a voltage to the task `bfieldCurrentOutputTask`. This is called in the method `button1_Click` found in the `ControlWindow` class. In the `ControlWindow.Designer`, Visual Studio automatically generates definitions for the various controllable objects. By default these are private, but if they are required by other classes, then they must be redefined as public objects. Before the application can launch, a `Runner` class must be written which contains the `Main()` sequence that initialises and starts the hardware controller. In here, it is also

possible to include remote access control so that the hardware controller can be controlled via other applications through TCP.

4.3 Including Vision Acquistion