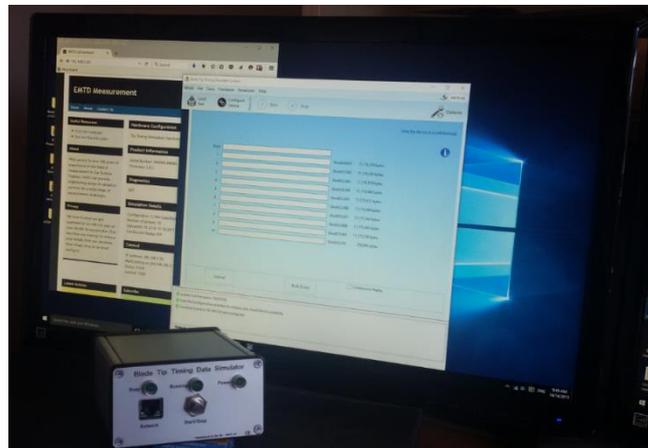


MultiTool Blade Tip Timing Acquisition, Analysis and Data Simulation Software

Injector Manual



Contents

1. MultiTool Injector Mode	3
1.1. Open Current Configuration Location	3
1.2. Loading a configuration file	3
1.3. Data Validation	4
1.4. Injector Display Sections	5
1.4.1. Speed History Chart (RPM) display (A+B)	6
1.4.2. Blade Impact (C)	7
1.4.3. Permanent Deformation (D)	8
1.4.4. Crack (E).....	9
2. Appendix 1 – Plug-in Development	10
Visual Studio Community 2017	10
2.1. Creating a new Project.....	10
2.2. Add A Reference to the Common Interface	12
2.3. Editing the PlugIn class.	12

Figures

Figure 1 - Switching to Injector mode.....	3
Figure 2 - Loading a configuration file	4
Figure 3 - Data Validation	4
Figure 4 - Injector Display Sections.....	5
Figure 5 – OPR History Display.....	6
Figure 6 –Blade Impact Settings.	7
Figure 7 – Permanent Deformation Settings	8

1. MultiTool Injector Mode

The Injector mode of MultiTool allows the user to adjust one or more blades in an actual dataset. This can be done either by injecting a permanent dc shift to the blades position, injecting an impact condition or propagating a crack that shifts the dc position of the blade with speed.

Some MultiTool features are licensed separately. If the Injector component of MultiTool has been licensed then it will be selectable from the Mode menu as shown in Figure 1. If this option is greyed and you believe that you have a valid license for Injector then please email support@emtd-measurement.com

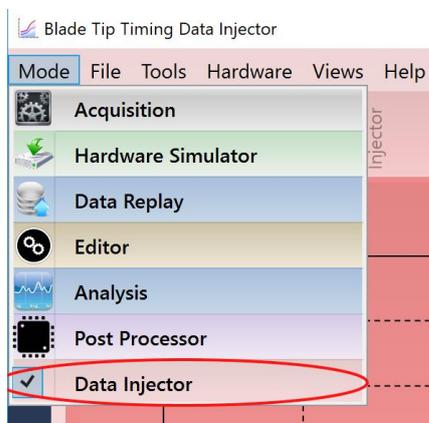


Figure 1 - Switching to Injector mode

1.1. Open Current Configuration Location

Once a configuration file has been loaded then this menu item will be enabled. It will open a new Windows Explorer window set to the current configuration files folder.

1.2. Loading a configuration file

In order to perform an injection a dataset a configuration file is required which describes the hardware configuration used to acquire the data. This must be a MultiTool standard configuration file and must be in the same directory as the data. If you don't already have one created then see Creating a Configuration file.

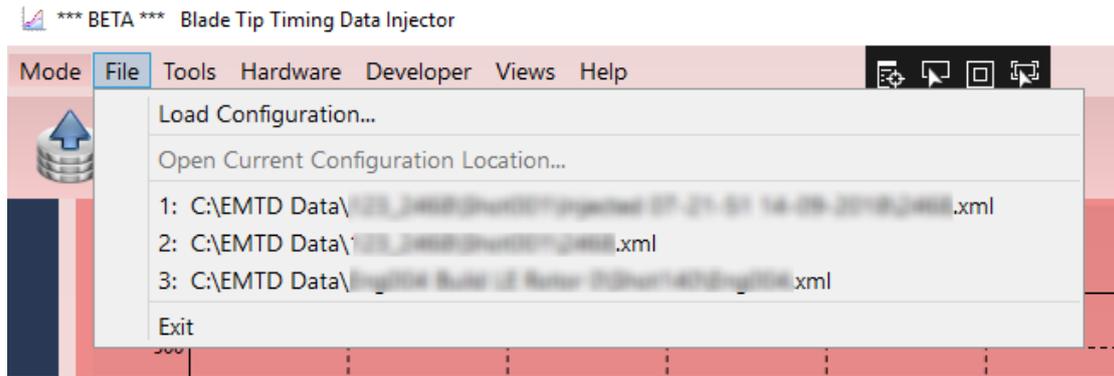


Figure 2 - Loading a configuration file

If the configuration file has been used before then it will appear in the recently used files list. Click on it to load. Note that the recent file list is different for each mode of the application so if you have loaded a configuration file in the editor it will appear in the recent list for the Editor but won't appear in the list for Injector until it has been loaded into the Injector system once.

If the configuration file has not been loaded before then select *Load Configuration* from menu.

1.3.Data Validation

MultiTool will inspect the dataset and make sure it is suitable for data injection. Any problems found will be reported in the status area at the bottom of the display. While some issues may be minor and are for information only any major issues will halt the process and must be fixed before the data can be used for injection. An example of a major issue would be the probe positions described in the configuration file do not match the positions calculated from the data. Once the data has passed the validation process we can proceed with Injecting an event(s) into it.

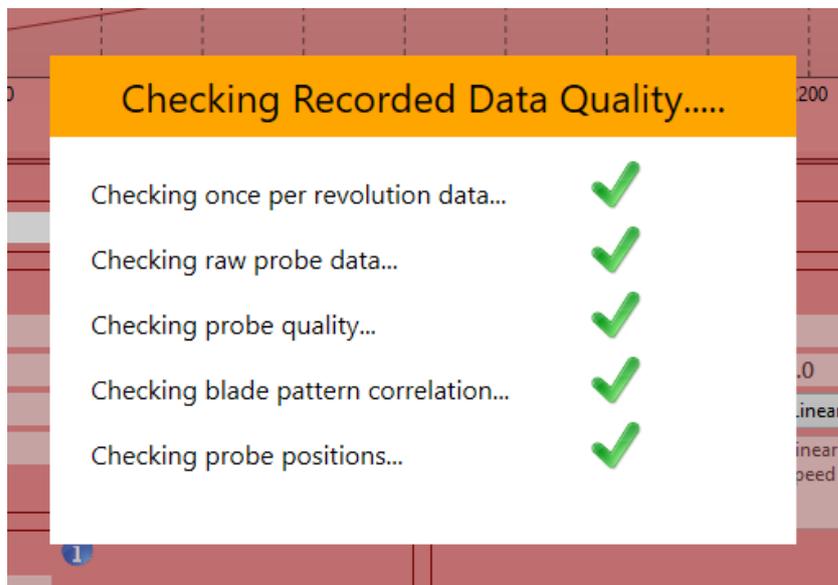


Figure 3 - Data Validation

Although MultiTool is fully multithreaded this can still take a little while depending on the size of the dataset and the capabilities of the PC being used.

Quick Load - Once a dataset has been successfully loaded once it will be marked as good. When this dataset is loaded again it will be quick loaded, bypassing the checking and speeding up the load process. Changing the configuration file will invalidate the quick load and cause a full check on the next load. If the dataset has data quality issues then the quick load will not be available until those issues are resolved. See relevant messages to aid in fixing any issues.

1.4. Injector Display Sections

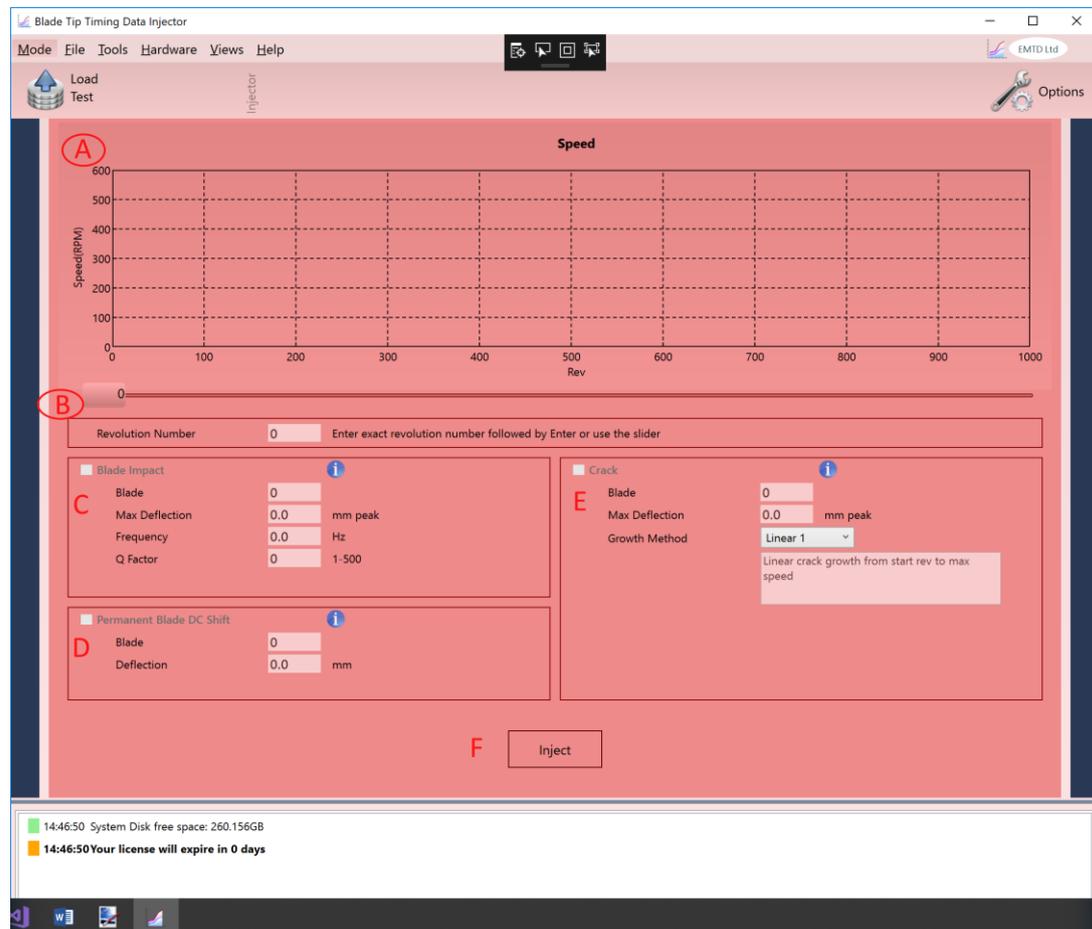


Figure 4 - Injector Display Sections

Once the dataset has been loaded the display will update similar to that shown in Figure 4 .



Extra help is available throughout MultiTool wherever there is an information icon. Hover the mouse over these icons to display useful information.

1.4.1. Speed History Chart (RPM) display (A+B)

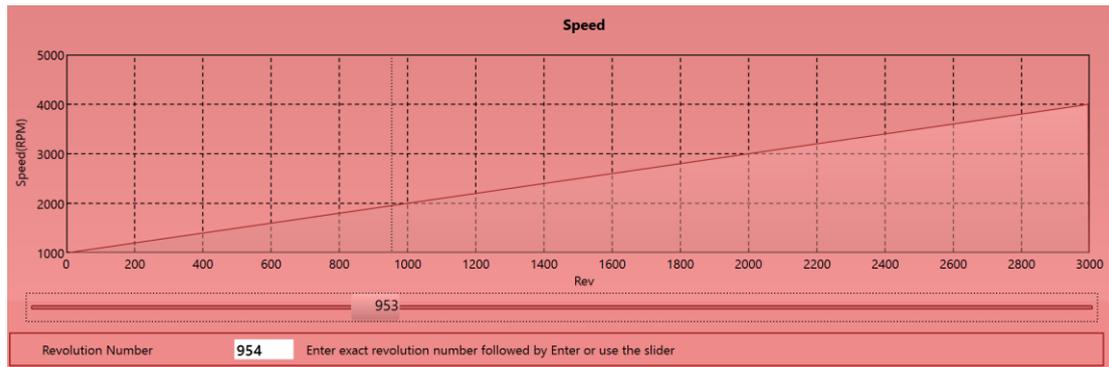


Figure 5 – OPR History Display

This display shows the speed history of the vehicle. The X axis is recorded revolutions which is effectively time. The Y Axis is in speed in Revolutions per Minute (RPM). The cursor underneath the display can be used to move the replay to a specific point in the file. For example if an event happened at a certain time. The current revolution number is shown on the cursor handle. To position the cursor exactly enter the desired revolution number and press the Enter key. Injected events generally begin their injection from this revolution onwards.



The display can be zoomed, and re-zoomed, with the left mouse button. To reset the display use the reset button on the bottom right of the display.

1.4.2. Blade Impact (C)

Blade Impact i

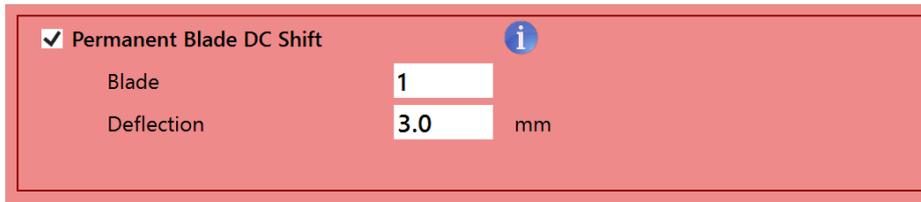
Blade	1	
Max Deflection	5.0	mm peak
Frequency	100.0	Hz
Q Factor	10	1-500

Figure 6 –Blade Impact Settings.

Blade Impact simulates an object hitting a blade. The impact will occur at the current revolution shown on the slider (**B**). The initial impact will move the blade's DC position to the value specified in Max Deflection where it will oscillate at the specified Frequency with a duration set by the Q Factor. The amplitude will decay at a rate defined by e^{-tB} .

Note:- This type of Injection can also be combined with the Permanent Deformation method.

1.4.3. Permanent Deformation (D)



The screenshot shows a settings panel with a red background. At the top left, there is a checked checkbox labeled "Permanent Blade DC Shift" and a blue information icon (i) to its right. Below this, there are two input fields: "Blade" with the value "1" and "Deflection" with the value "3.0" followed by the unit "mm".

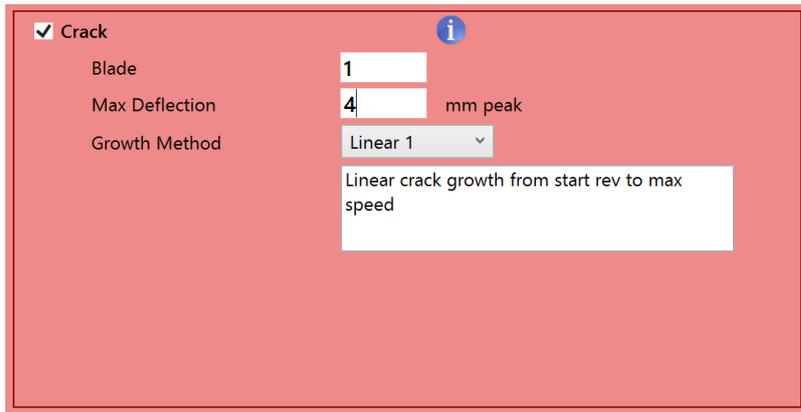
Figure 7 – Permanent Deformation Settings

A blade can undergo a Permanent Deformation for several reasons, one of which is the Blade Impact injection above. This will instantly move the blade's DC position by the Deflection amount at the selected revolution, lasting for the remainder of the file.

Note:- When viewing this DC shift in the Replay mode's View Stack function and sliding the slider through the file, the shift will not appear instantly. This is because the Replay's View Stack averages a block of revolutions to obtain a cleaner picture. Once this average contains all of the shifted data then the blade will be shown correctly at the desired DC shift. This is a function of the display and not the data.

When combined with the Blade Impact Injection this can produce an impact that leaves a residual DC shift on the blade. If the start revolution is the same for both methods then the Blade Impact oscillation will start at the DC offset plus the Max Deflection value and decay to the DC offset value.

1.4.4. Crack (E)



The screenshot shows a configuration window for the 'Crack' mode. It features a red background and a white border. At the top left, there is a checked checkbox labeled 'Crack' and an information icon. Below this, there are three settings: 'Blade' with a value of '1', 'Max Deflection' with a value of '4' and the unit 'mm peak', and 'Growth Method' with a dropdown menu set to 'Linear 1'. A tooltip is visible over the dropdown menu, containing the text: 'Linear crack growth from start rev to max speed'.

The Crack Injection method cannot be used in conjunction with the other two modes in the same injection. However, multiple injections are possible on the same file so can be added separately. This mode allows the DC position of a blade to vary as a function of speed. As shown above the crack will result in a shift in the blade's DC position which will be 4mm at the maximum speed defined in the configuration file. The crack will start at the current revolution number unless stated otherwise.

The crack growth algorithm can be selected from the drop-down list. In this case the algorithm is Linear 1. This will grow the crack at a constant rate between the current revolution number set by the slider (**B**) and the max speed set in the configuration file. The configuration file maximum speed is used so that the same affect can be applied to several datasets with the same result. The Max Deflection DC value will be found only in datasets that reach that speed.

Injector supports a plug-in system whereas the user can specify the crack algorithm used on the blades DC position, either by selecting a predefined one or by creating their own plug-in. Details of how to do this are shown in Appendix 1.

2. Appendix 1 – Plug-in Development

In order to create a plug-in for Injector you will need a copy of the free Microsoft product:

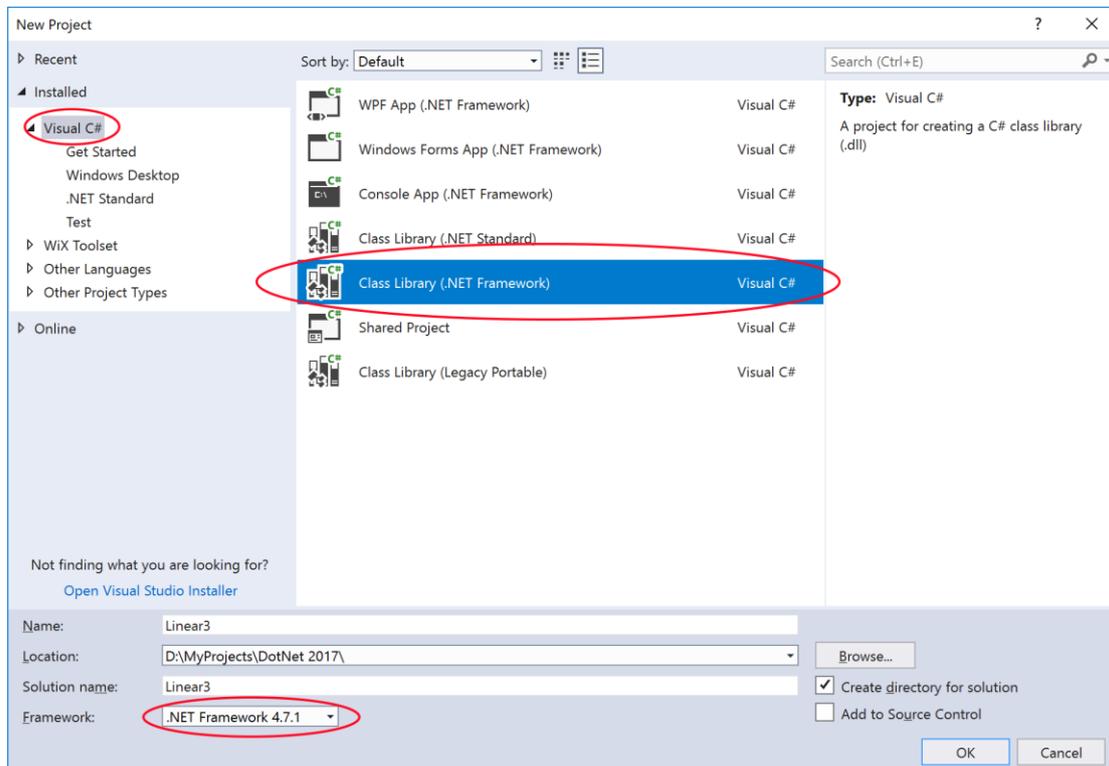
[Visual Studio Community 2017](#)

If you have the professional or enterprise version you do not need to download the free edition.

2.1. Creating a new Project

Note:- A sample project is available containing the source code used for Linear1.

In Visual Studio 2017 menus, select File->New Project and you will see the following window.



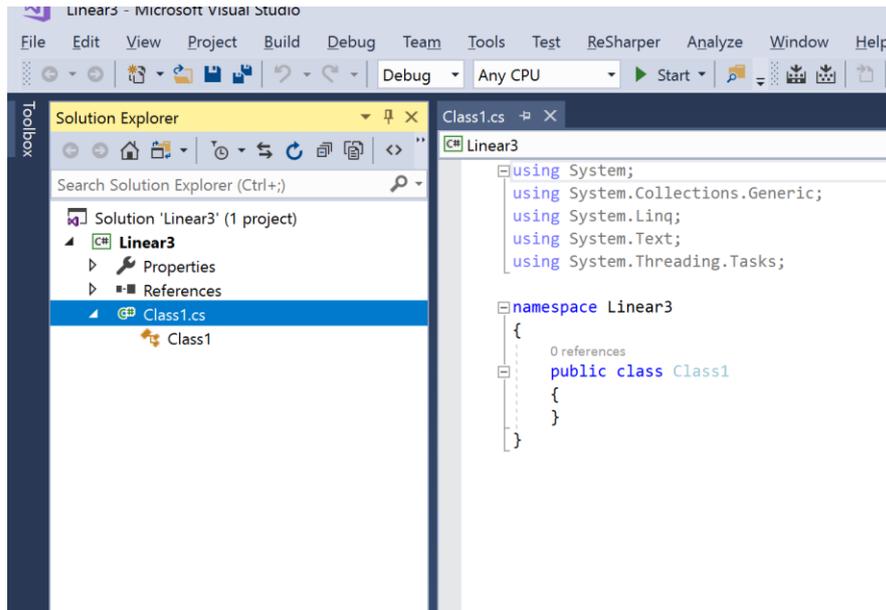
You should call the project something relevant, in this case Linear3.

Make sure to select the correct framework version as shown.

Make sure to select the correct project type as shown.

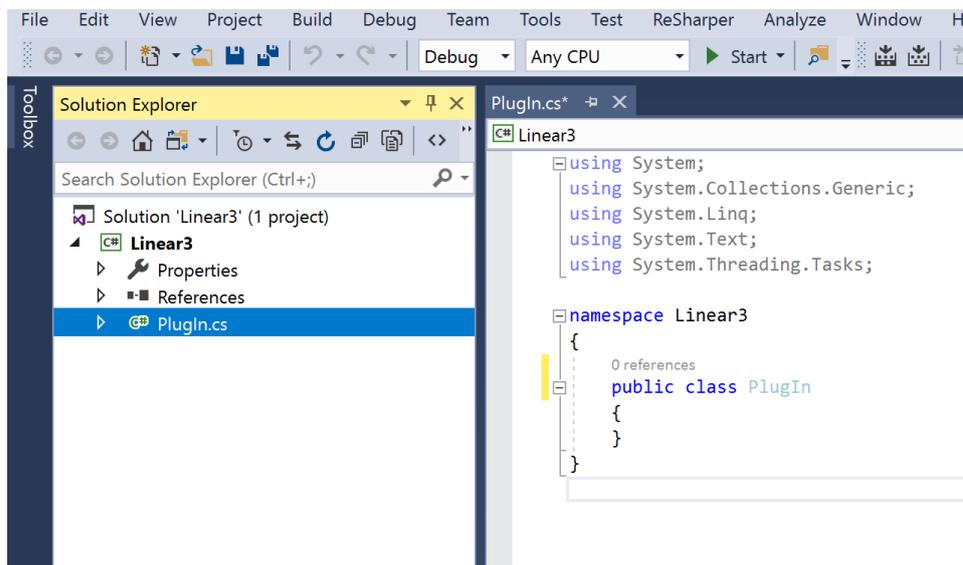
When the correct settings have been chosen then press OK to create the new project.

In the solution Explorer you should see the following when the file class1 is highlighted.



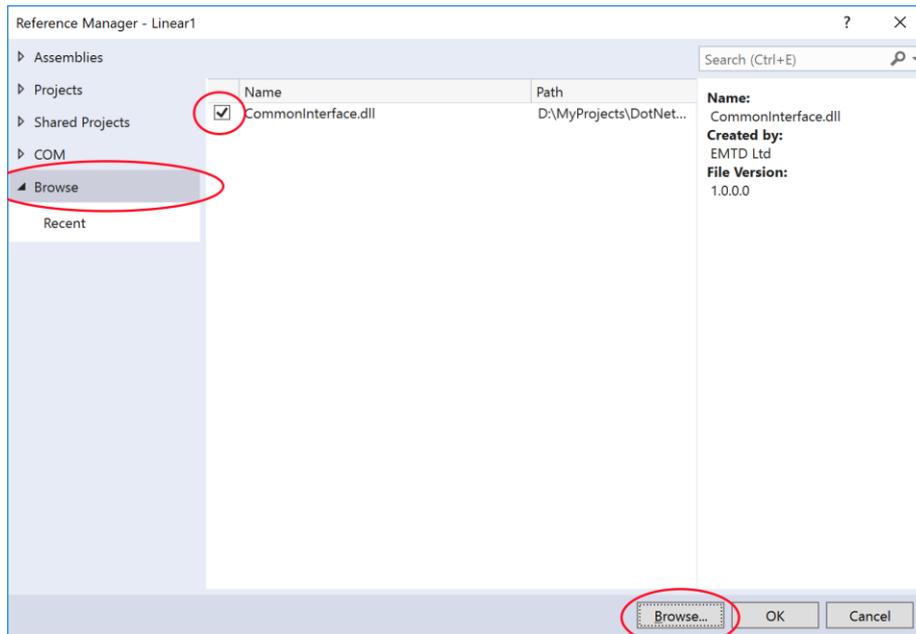
Right click on the Class1.cs entry (Highlighted) and select Rename. Change the name of this class to PlugIn. Note the capital and I and no spaces.

Visual Studio Will ask you if you want to rename the class and any instances. Select OK. You now have a plug-in file ready for some code.



Note:- It is essential that the PlugIn class has the correct spelling and that the namespace (Linear3) matches the project namespace. When copying over the PlugIn class don't copy (and change) the namespace line.

2.2. Add A Reference to the Common Interface



In Solution Explorer (see above) right click on the references and select add reference. You should see the above dialog. Select Browse on the left panel and browse at the bottom to the installation folder. You want to add a file called CommonInterface.dll. Select it, tick the box as shown above and add it to the project.

2.3. Editing the PlugIn class.

You can either copy the PlugIn.cs file from the Linear1 example or create your own from the below code snippet.

The Calculate routine will be called by Injector when a new value for the DC offset is required. The returned value should be normalised to a range of 0->1. A value of 0 will have no effect on the dc position of the blade and value of 1 will apply the MaxDeflection value.

The user is free to do whatever they like with the value, returned values outside of this range will be set to either 0 or 1 depending on the value.

Select Release version and compile the project. If there are any errors then these will need to be fixed first.

The resulting dll file can then be copied to the PlugIn directory in the install folder and Injector will add it to the list of available algorithms.

Note:- The list of algorithms is updated when MultiTool switches to Injector mode. To refresh the list switch to another mode and back again.

The default folder for plugins is C:\EMTD\Bin\PlugIns

```
[Serializable]
public class PlugIn : MarshalByRefObject, IPlugIn
{
    public string Name { get; private set; }
    public string Description { get; set; }
    public Double StartSpeed { get; set; }
    public Double StopSpeed { get; set; }
    public Double MaxSpeed { get; set; }
    public Double MaxSpeedInFile { get; set; }

    public void Initialize()
    {
        Name = "Linear 1";
        Description = "Linear crack growth from start rev to max speed";
    }

    /// <summary>
    /// Return a normalized value for  $Y = F(x)$  amplitude varies between 0 and 1
    /// </summary>
    public double Calculate( double speed)
    {
        var range = MaxSpeed - StartSpeed;
        if (range < 0) return 0;

        var value = (speed - StartSpeed) / range;

        return value;
    }

    public void PrintLoadedAssemblies()
    {
        Helpers.PrintLoadedAssemblies();
    }
}
```