Distributed Control and Automation Framework Hands-On Session

This hands-on covers the basics of implementing an application in the Distributed Control and Automation Framework, including using an existing module and developing a new control module. It doesn’t cover development of a new generic I/O or processing module. For this hands on, the framework downloads, and additional documentation, visit <https://decibel.ni.com/content/groups/dcaf> .

## Set-up (Already configured on Alliance Day machines)

Navigate to <https://decibel.ni.com/content/docs/DOC-41727> and download the .vipc file attached to that page. This package can be installed to any LabVIEW version from 2013 to present.

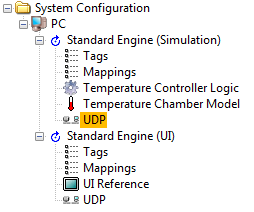
# Exercise 1: Configuring a Simple Temperature Control Application

**Introduction**

This exercise demonstrates the implementation of a simple temperature chamber controller application using the Distributed Control and Automation Framework (DCAF). This exercise makes use of a model of the chamber to simulate its I/O and allows users to define the setpoint and PID gains of the control algorithm through a simple user interface.

During the exercise you will learn to identify how inputs and outputs from different modules are mapped into the DCAF Tag Bus to provide a synchronized communication between modules. You will also learn how to create a UI and map it to the Tag Bus.

Our Temperature Simulated Controller will consist of 2 DCAF engines: the UI and the Temperature Controller Simulation.



*Figure 1.1*

This Simulation Engine is implemented in such a way that it will be easy to migrate from a simulation to a real time controller by adding a cRIO target and moving the Simulation Engine below it.

Notice the different components of each engine listed in Fig 1.1. Tags, Mappings and UDP repeat in both engines. The Temperature Controller Logic and the Temperature Chamber Model are Modules created specifically for the Simulation Engine. Similarly, the UI Reference Module is only present in the UI Engine. Here is a brief description of each of these components.

**Tags:** Tags are scalar variables that can be mapped as inputs or outputs in any module within a DCAF engine. This pane allows the user to add or remove tags to the engine and configure tag properties such as Tag Name, Data Type, Default Value, and Description.

**Mappings:** This pane allows the user to connect tags as inputs or outputs in DCAF modules. For example, an Output Channel from the Temperature Controller Logic Module can be mapped into a Tag which can be mapped as an Input Channel to the Temperature Chamber Model Module.

**UDP:** This pane is also present in both components. It is designed to share tags between Engines by mapping each tag as an Engine Input or Output. All the tags that are intended to be shared between engines need to be defined in the Tags Pane of each engine with the same names.

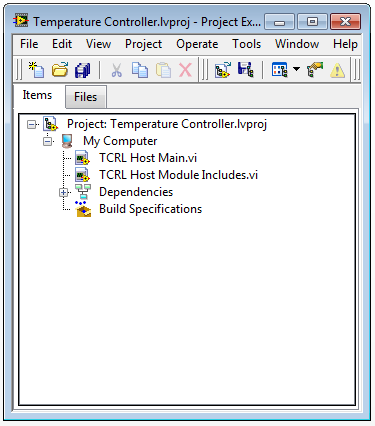
**UI Reference:** This module only works in the UI Engine. It provides a front panel with the necessary controls and indicators to allow the user interact with the DCAF Engine.

**Temperature Controller Logic:** This is a custom DCAF Model designed to provide a Temperature Control Logic. If the Simulation Engine is moved to a cRIO Target and the Temperature Chamber Model is replaced with real IO, this module could remain the same.

**Temperature Chamber Model:** This module provides a simulated model of a Temperature Chamber. This module could be replaced or overwritten to eventually provide IO from a real Temperature Chamber.

**Part 1: Project Creation and UI**

1. In LabVIEW go to **File >> Create Project..** and select TBDF. From the displayed list select **Basic Execution Template** and press the ***Next*** button.
2. Name the project **Temperature Controller** and select [**\\Desktop\DCAF Hands On\Exercise 1**](file:///\\Desktop\DCAF%20Hands%20On\Temperature%20Controller\Exercise%201)**\Temperature Controller\Runtime** as the Project Root. Type **TCRL** as the File Name Prefix.
3. Verify your project window looks like Figure 1.2.

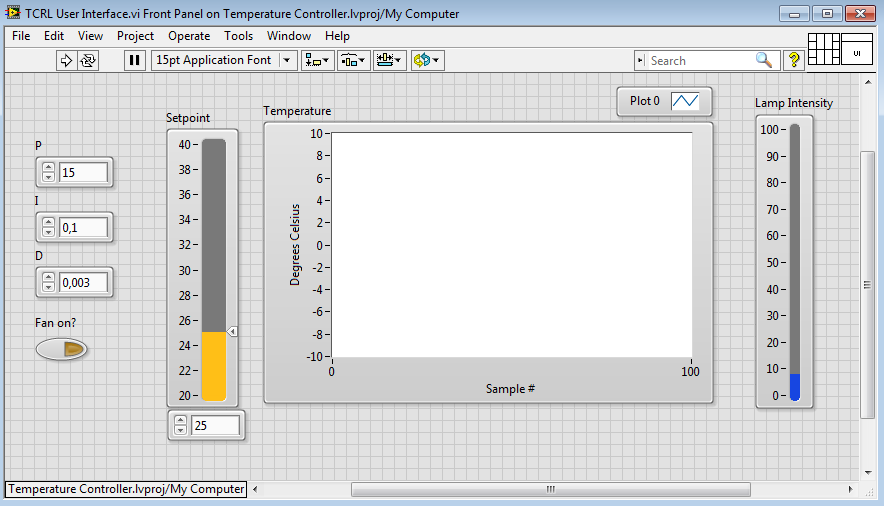


*Figure 1.2*

1. Add to the project a Configuration File with the Engines for the Temperature Controller partially configured and mapped. Under **My Computer** add ***SimulatedSystem.pcfg***  located at **\\Desktop\DCAF Hands On\Exercise 1\ Temperature Controller**.
2. Create a UI for our Temperature Controller. Add a New VI to the project and save it as ***TCRL User Interface*** in **\\Desktop\DCAF Hands On\Exercise 1\Temperature Controller\Runtime.**
3. Add to the Front Panel the following Controls and Indicators**:**

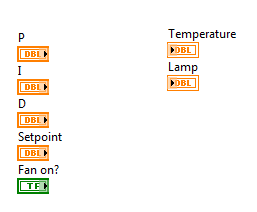
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Control Type** | **Data Type** | **Label** | **Caption** | **Default Value** | **Notes** |
| Numeric Control (Silver) | Double | P | P | 15 |  |
| Numeric Control (Silver) | Double | I | I | 0,1 |  |
| Numeric Control (Silver) | Double | D | D | 0,003 |  |
| Vertical Pointer Slide (Silver) | Double | Setpoint | Setpoint | 25 | Visible Items >> Digital Display. Range 20 – 40. Change to indicator. |
| Push Button (Silver) | Boolean | Fan on? | Fan On? | False |  |
| Waveform Chart (Silver) | Double | Temperature | Temperature | 20 | Y Axis: Degrees Celsius. X Axis: Sample # |
| Vertical Fill Slide (Silver) | Double | Lamp | Lamp Intensity |  | Range: 0 - 100 |

1. Verify your UI looks like Figure 1.3.



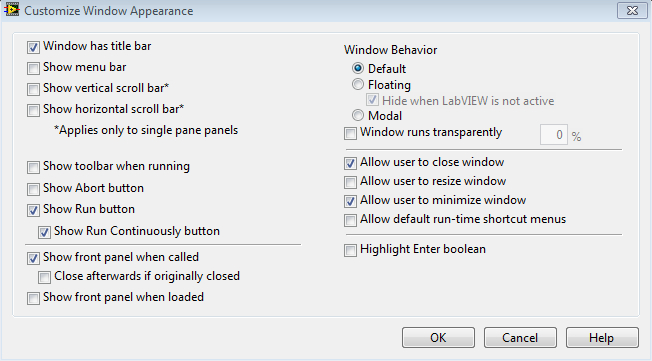
*Figure 1.3*

1. Verify the names of the Labels in the Block Diagram. This is important to correctly map the tags to the UI. Controls and Indicators will be directly updated through the DCAF UI Engine, so there is no need to add more code in this VI.



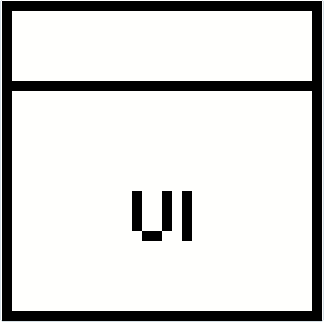
*Figure 1.4*

1. Adjust the Front Panel window size and configure this VI Window Appearance (File >> Vi Properties >> Window Appearance). Select ***Custom*** and press the ***Customize..*** Button. Configure the Window Appearance as appears in the next image.

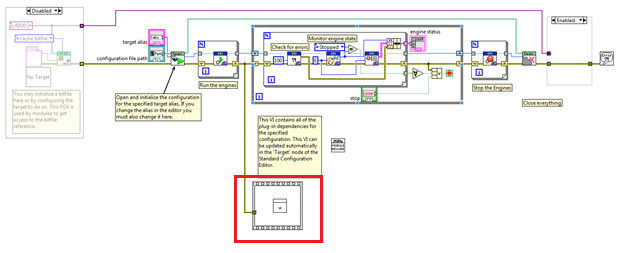


*Figure 1.5*

1. Edit your Icon in a very simple way.



1. Save and Close TCRL User Interface.vi.
2. Open **TCRL Host Main.vi** Block Diagram. Drag and Drop **TCRL User Interface.vi** into the Block Diagram from the Project Window. Force TCRL User Interface.vi to execute in parallel to the DCAF engine connecting as appears in Figure 1.6.

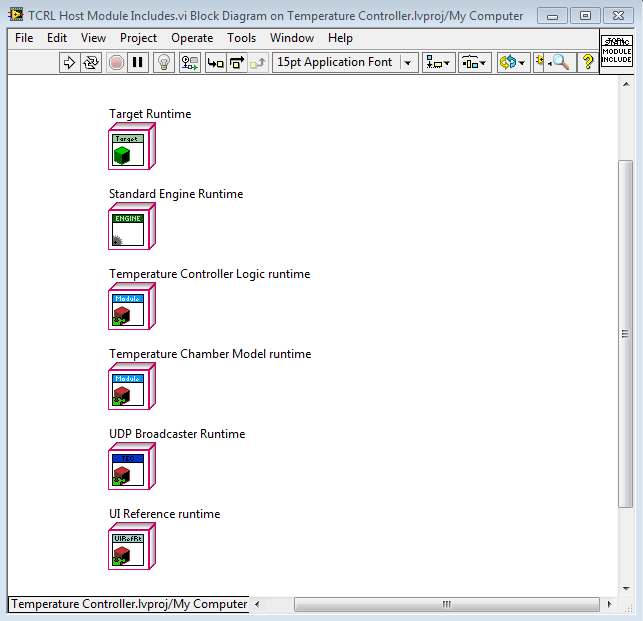
****

*Figure 1.6*

1. Open TCRL Host Main.vi Front Panel. In the ***configuration file path control*** browse for ***SymulatedSystem.pcfg*** located at [**\\Desktop\DCAF Hands On\Exercise 1**](file:///\\Desktop\DCAF%20Hands%20On\Temperature%20Controller\Exercise%201)\**Temperature Controller** the configuration file path. Select this as default value for this control. Save and Close this VI.

**Part 2: Adding Required Classes**

1. Open the Standard Configuration Editor for TBD by navigating in LabVIEW to **Tools>>TBDF>>Launch Standard Configuration Editor…**
2. Navigate within the editor to **Tools>>Edit Plugin Search Paths**.
3. Add a search path to the TBM plugins for this example located at **\\Desktop\DCAF Hands On\ \Exercise 1\Temperature Controller** if it’s not already there.
4. In the DCAF Configuration Editor go to **File>>Open** and search for the SimulatedSystem.pcfg Configuration File located **at** [**\\Desktop\DCAF Hands On\Exercise 1**](file:///\\Desktop\DCAF%20Hands%20On\Temperature%20Controller\Exercise%201)\**Temperature Controller**.
5. Take a couple of minutes to go through each component in the Simulation and UI Engines.
6. Open the Temperature Controller project located at [**\\Desktop\DCAF Hands On\Temperature Controller\Exercise 1**](file:///\\Desktop\DCAF%20Hands%20On\Temperature%20Controller\Exercise%201)**.**
7. Open **TCRL Host Module Includes.vi** and verify the Block Diagram is empty. This VI will load the required classes when **TCRL Host Main.vi** executes. Use the Scripting Tool to add the corresponding classes to **TCRL Host Module Includes.vi**. To use the Scripting Tool go back to the **Configuration Editor** and select ***Tools >> Script Include VI***. **Select TCRL Host Module Includes.vi** located at [**\\Desktop\DCAF Hands On\Excercise**](file:///\\Desktop\DCAF%20Hands%20On\Temperature%20Controller\Exercise%201) **1\Temperature Controller\Runtime**. Select ***PC*** when the ***Enter Target to Script*** window appears. Verify the TCRL Host Module Includes.vi looks like Figure 1.7:

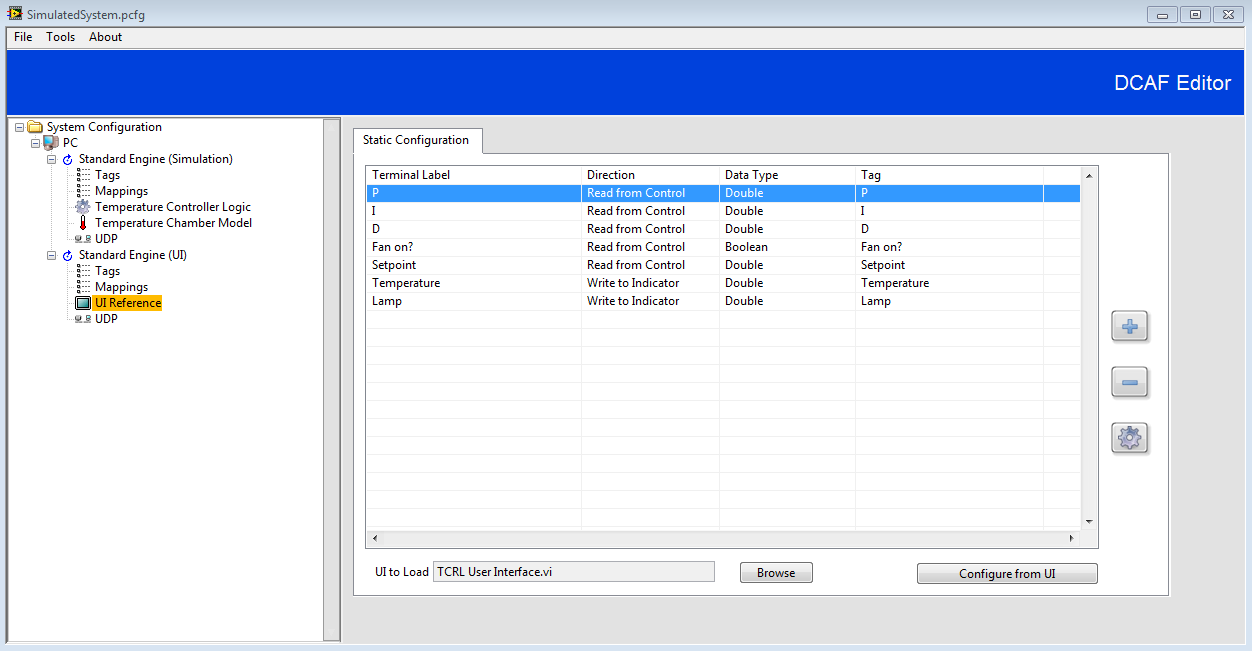


*Figure 1.7*

1. Save and close **TCRL Host Module Includes.vi**.

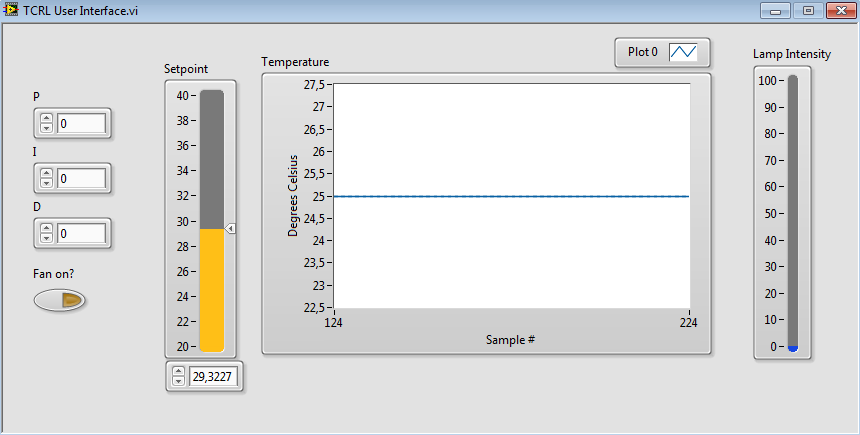
**Part 3: Mapping Tags in the Configuration Editor**

1. Open the Standard Configuration Editor for TBD by navigating in LabVIEW to **Tools>>TBDF>>Launch Standard Configuration Editor…**
2. We will first map our UI to the UI Engine Tags. Beneath the UI Standard Engine select UI Reference. Notice the table in the Static Configuration tab is empty. Press the ***Browse*** button next to the ***UI to Load*** textbox. Select the UI you created (***TCRL User Interface.vi***) located at [**\\Desktop\DCAF Hands On\Temperature Controller\Exercise 1**](file:///\\Desktop\DCAF%20Hands%20On\Temperature%20Controller\Exercise%201)**\Runtime.**
3. Press the ***Configure from UI button***. When the pop up asking to ***Automatically map tags to channels*** appears select ***Yes***. Verify your mapping comparing it with Figure 1.8.



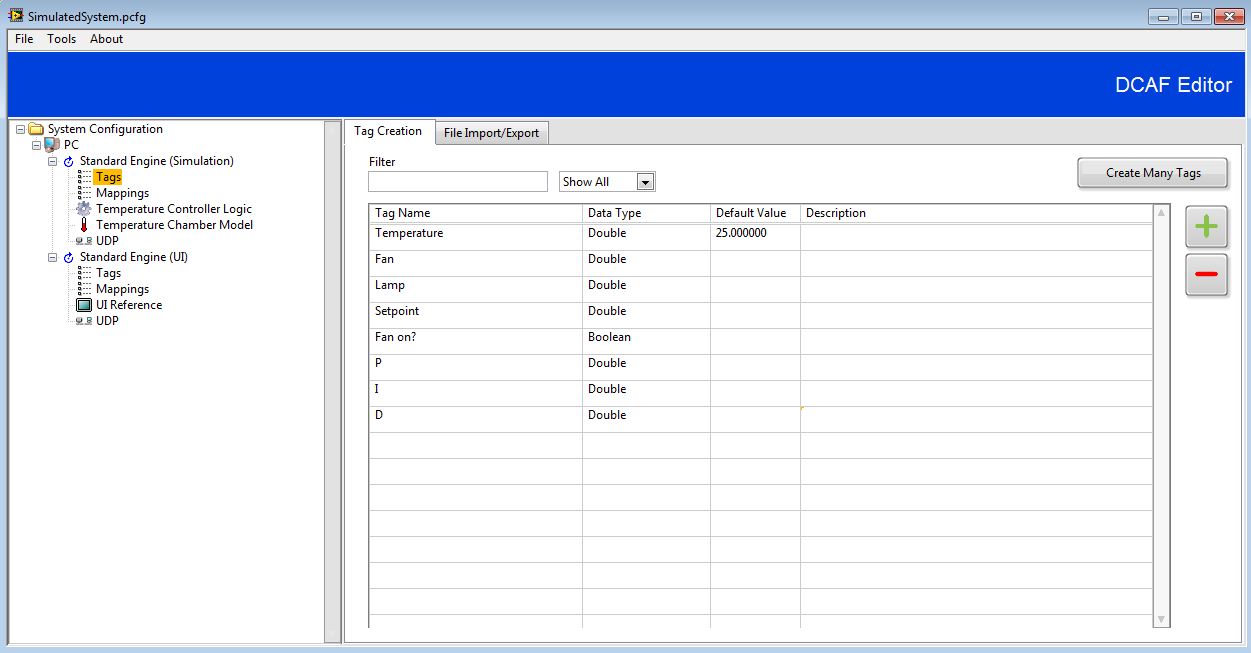
*Figure 1.8*

1. Save the changes in the Configuration Editor.
2. Open and run **Host Main.vi**. Try changing the Setpoint and the other controls. Do you see any change in the temperature value displayed in the Graph?



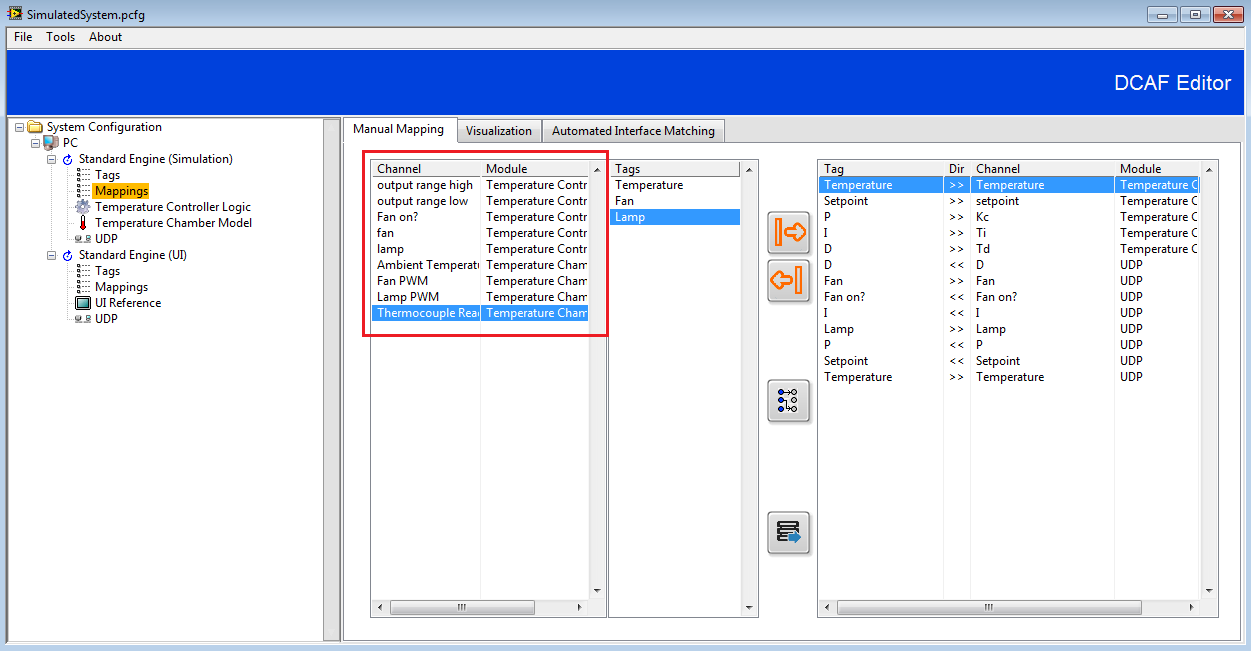
*Figure 1.9*

1. You shouldn’t see any change in the signal since we only connected the tags in the UI Engine. There are still some tags in the Simulation Engine that we need to map so we can see the PID standard behavior.
2. Stop the VI and return to the Configuration Editor. We will review the connections in each component on both engines to understand the tag dataflow and connect the tags that are missing to make it run.
3. We will start with the Simulation Engine. First select the Tags component and take a look to the tags.



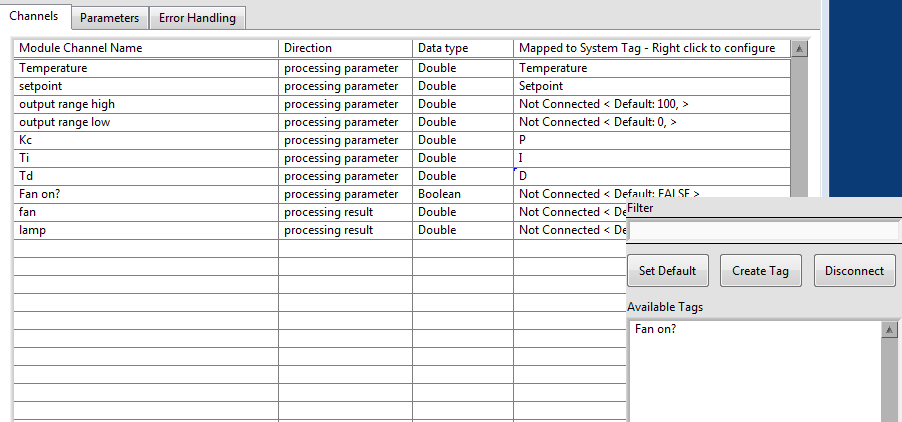
*Figure 1.10*

1. This tags are used as inputs or outputs in the rest of Simulation Engine modules: Temperature Controller Logic, Temperature Chamber Model and UDP. Notice all of them are Doubles except for ***Fan on?****.*
2. Go to ***Mappings*** and select the ***Manual Mapping*** tab. This section will allow you to have a better look of the tag flow in this application. In the left pane you will see all the channels that haven’t been mapped. Just take a look, don’t make changes.



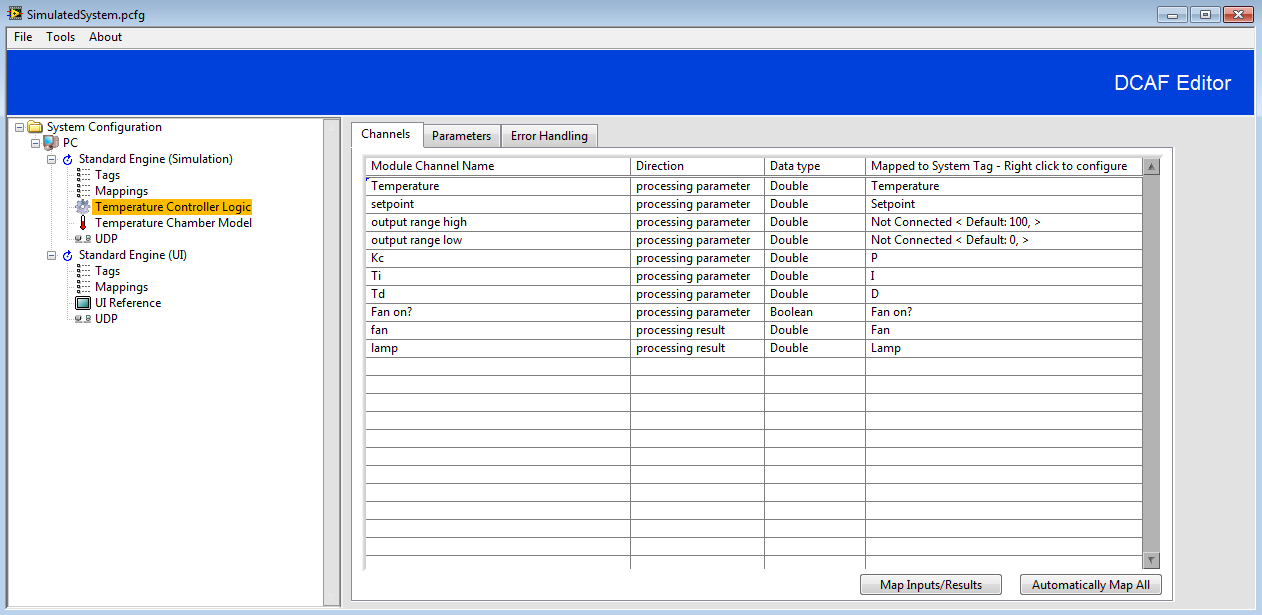
*Figure 1.11*

1. Go to the Temperature Controller Logic Module. Notice there are two variables that don’t appear in the Tag list: ***output range high*** and ***output range low***. This are internal variables with constant values defined statically. The rest should be mapped to a tag.
2. The last 3 channels should be connected to a tag (***Fan on?***, ***fan***, and ***lamp***). To connect a channel to a tag, take the cursor to the corresponding cell in the ***Mapped to System Tag*** column, right click, and select the corresponding tag from the ***Available Tags*** list.



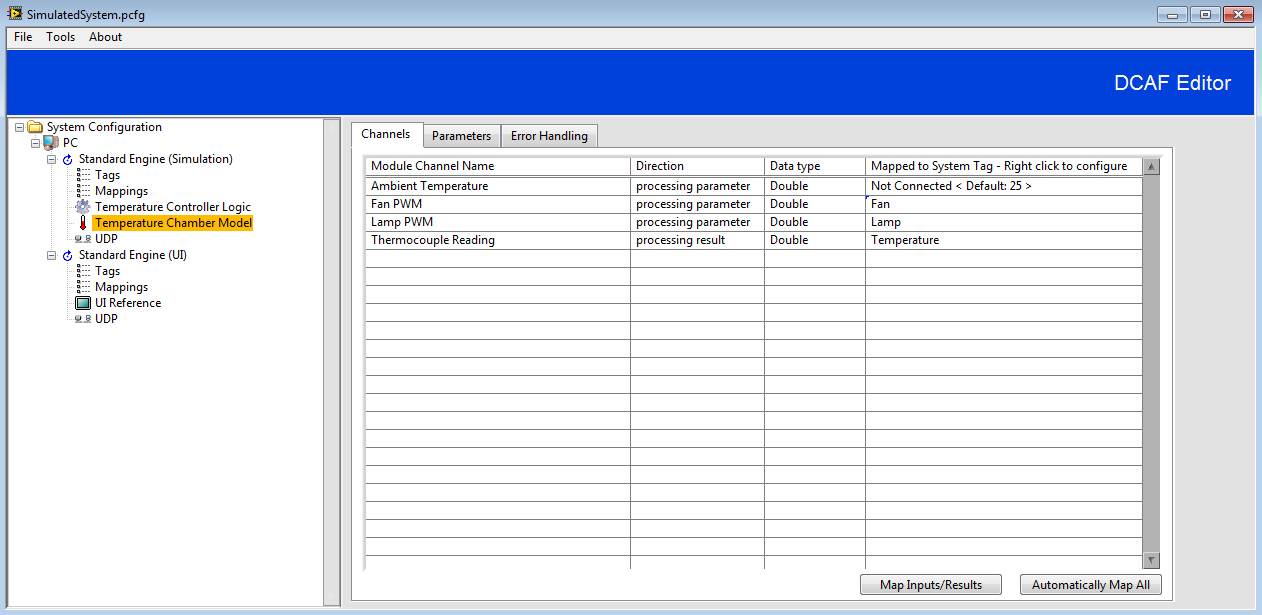
*Figure 1.12*

1. Verify your table looks like the following image:



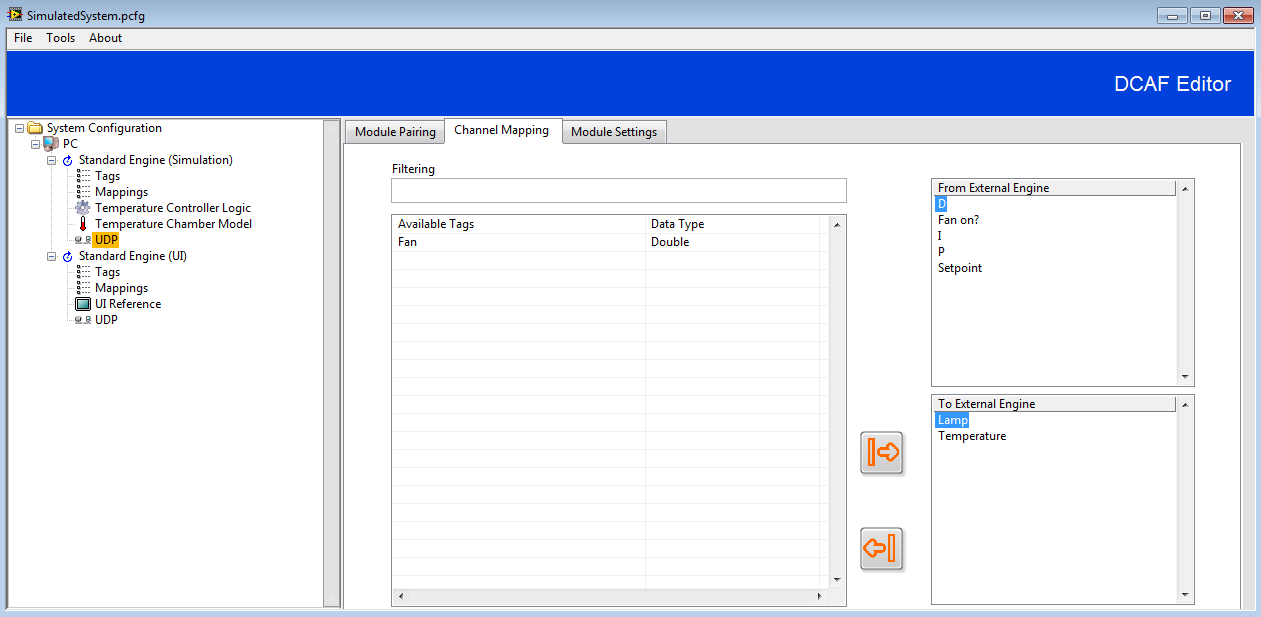
*Figure 1.13*

1. Before going to the next module notice the ***Direction*** column. ***Processing parameters*** are module inputs while ***processing results*** are module outputs. Some of the ***processing parameters*** in this module come from the ***UI Engine*** and others come from the ***Temperature Controller Logic Module***. The two ***processing results*** in this module will go through the ***Tag Bus*** as inputs in the ***Temperature Chamber Model Module***.
2. Go to the ***Temperature Chamber Model Module***. Notice all the channels are disconnected from any tag. The only disconnected channel should be ***Ambient Temperature***. ***Fan PWM*** and ***Lamp PWM*** channels are ***processing parameters*** in this module that should come from *the* ***Temperature Controller Logic Module***. ***Thermocouple Reading*** is a ***processing result*** that should be used as the feedback signal in the ***Temperature Controller Logic Module*** and will also be sent to the ***UI Engine*** to be displayed in the graph. Following the same instructions as in step 14, map ***Fan PWM***, ***Lamp PWM***, and ***Thermocouple Reading*** channels to ***Fan***, ***Lamp***, and ***Thermocouple*** tags. Verify your table looks like the following image:



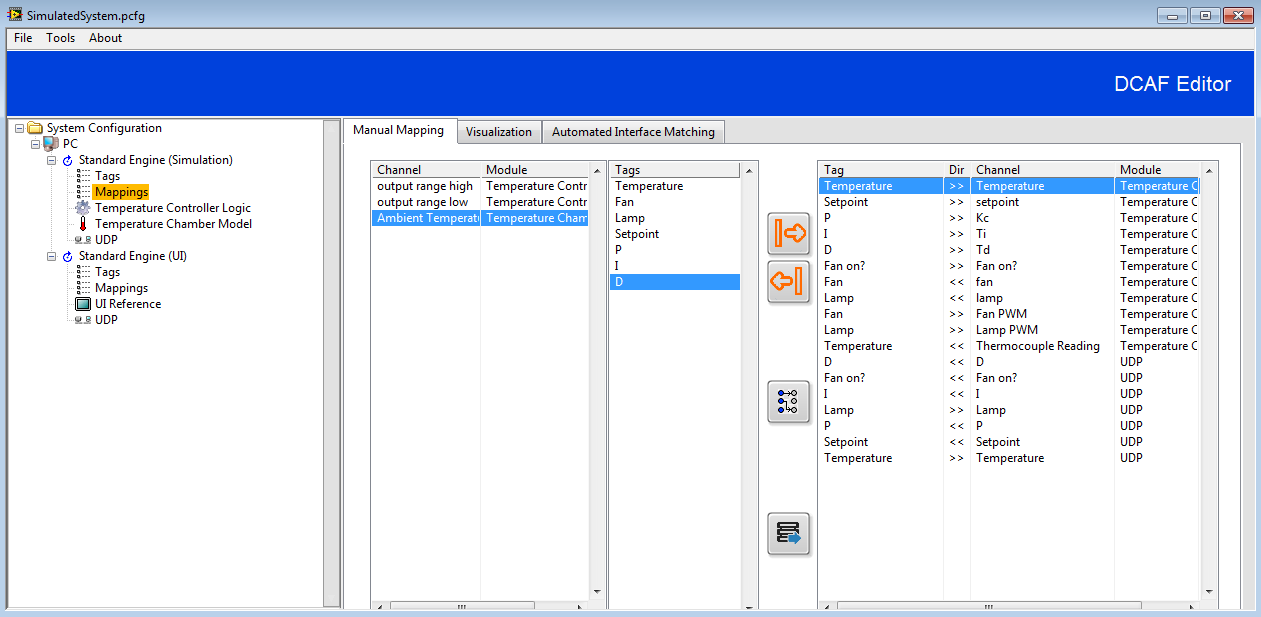
*Figure 1.14*

1. Go to the ***UDP Module*** in the ***Simulation Engine***. Go to the ***Channel Mapping Tab***. Notice the tags in the ***From External Engine*** (Inputs) and ***To External Engine*** (Outputs) boxes. Notice the ***Fan*** tag is still as an ***Available Tag***. There is no need to move it since it is not needed in the ***UI Engine,*** it is only used internally in the ***Simulation Engine.***



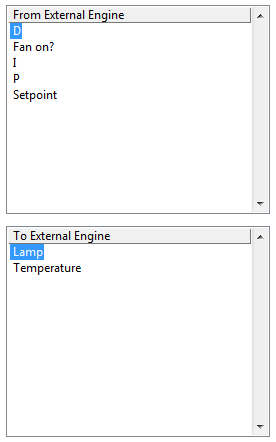
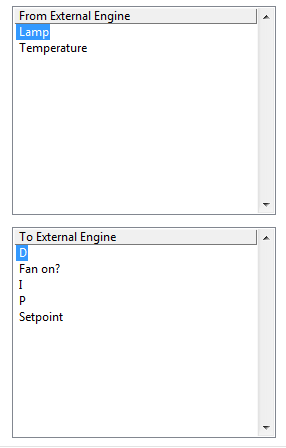
*Figure 1.15*

1. Go back to ***Mappings*** in the ***Simulation Engine***. Notice now there are only 3 channels that haven’t been mapped. There are no tags for those channels since they are configured statically in their corresponding modules or set as default. All the channels that originally were unmapped now appear mapped in the right pane. Take some time to review the mapping directions to have a better understanding of the data flow.



*Figure 1.16*

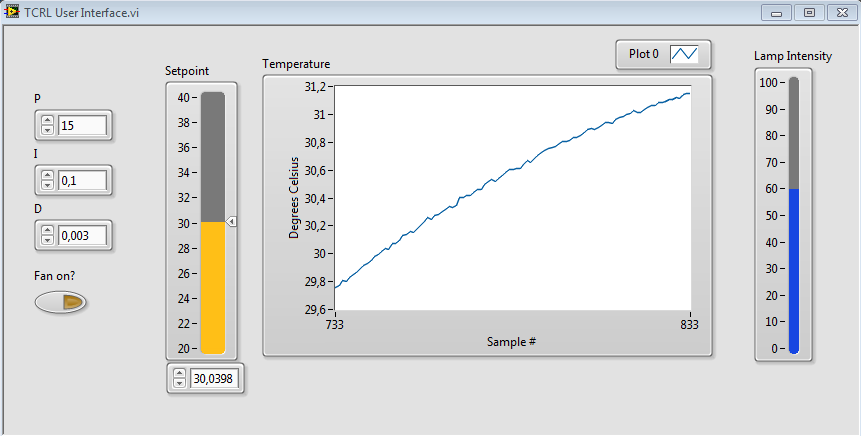
1. Now you have understood how the ***Simulation Engine*** works, step into the different components of the ***UI Engine*** to understand how it interacts with the ***Simulation Engine***. Notice that the Inputs for the ***UI Engine UDP Module*** are the Outputs for the ***Simulation Engine UDP Module*** and viceversa.

Simulation Engine UDP UI Engine UDP

*Figure 1.17*

1. Go to ***File >> Save*** and close the ***Configuration Editor***.
2. Open the ***Temperature Controller Example Project*** if not already open. Open and run ***Host Main.vi***.
3. Modify the ***Setpoint*** and the other controls in the ***UI***. You should now see the temperature being controlled by the ***Simulation Engine***.



*Figure 1.18*

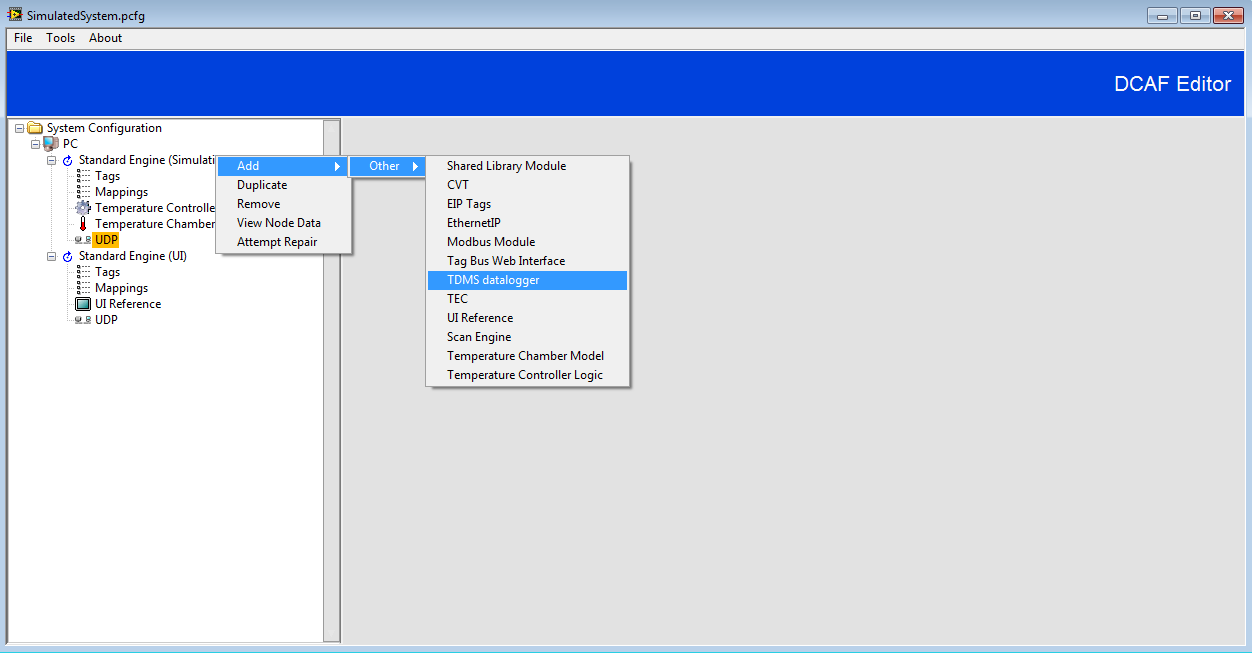
***End of Exercise 1***

# Exercise 2: Adding Standard Modules to the Temperature Control Application (TDMS & CVT)

In Exercise 1 you developed a Simple Temperature Control Application using DCAF. We are going to add standard features such as TDMS and CVT to learn how to add standard DCAF modules to your application.

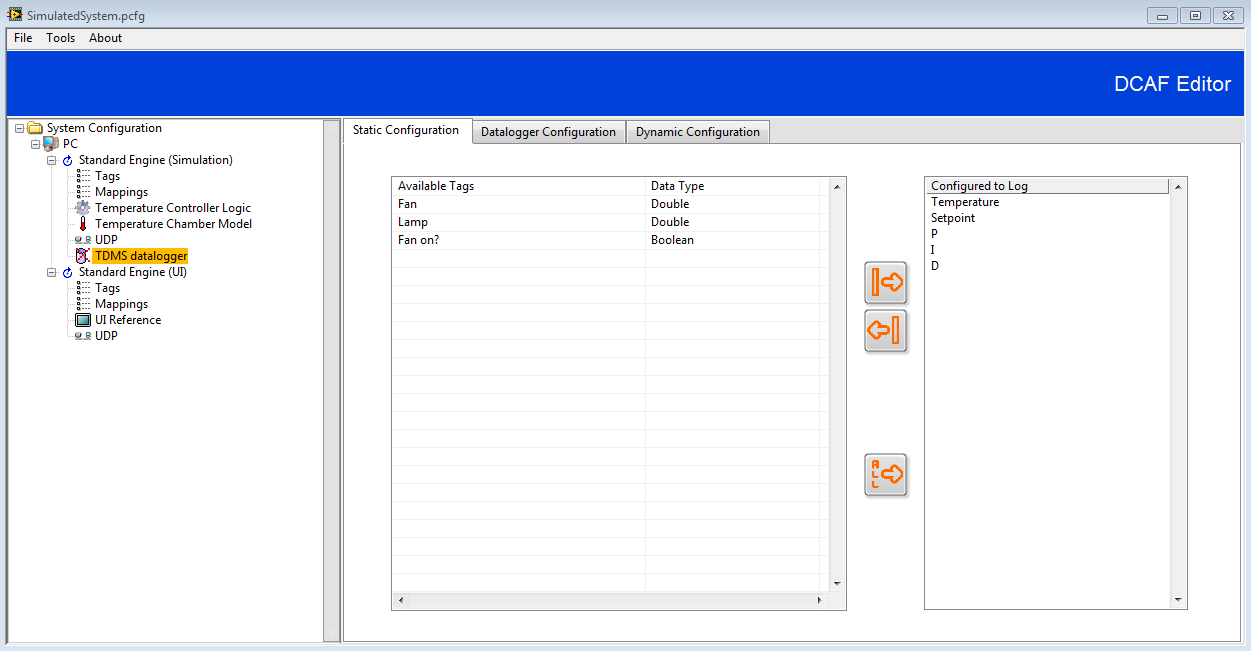
**Part 1: Add TDMS**

1. Open the ***Temperature Controller*** project you developed in Exercise 1 if not already opened.
2. Open the ***Configuration Editor*** and load ***SimulatedSystem.pcfg*** if not already opened.
3. Right click the ***Simulation Engine*** and select ***Add>>Other>>TDMS datalogger*** as shown in Figure 2.1.



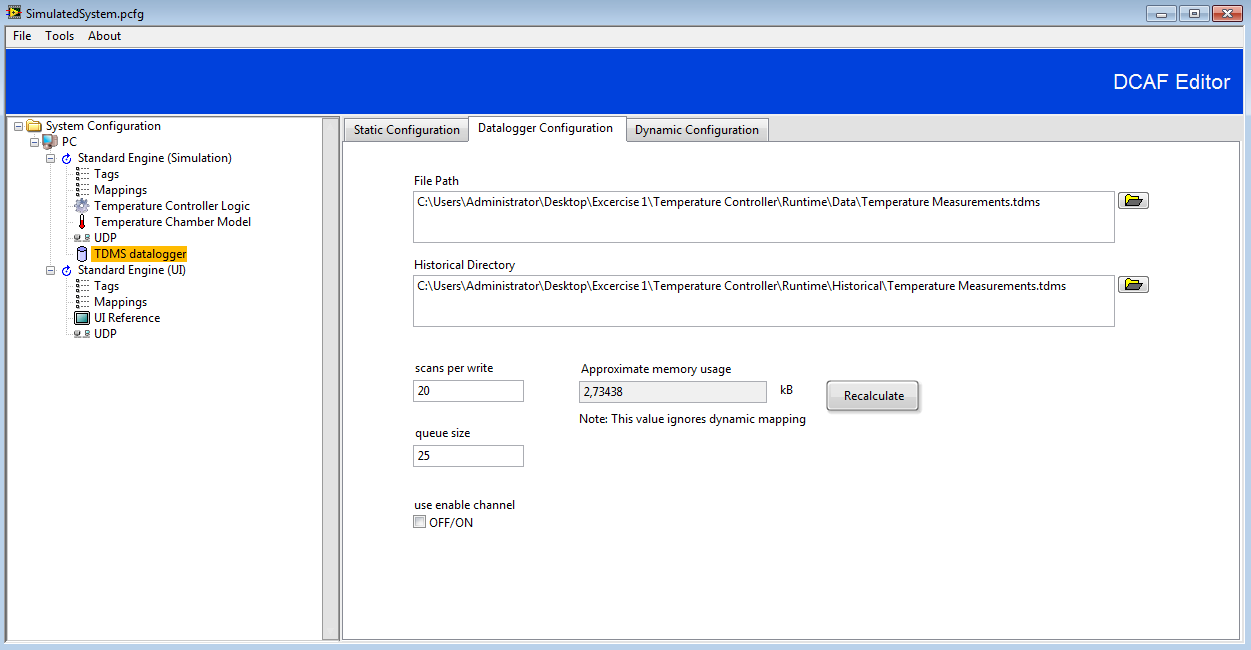
*Figure 2.1*

1. In the Static Configuration tab move Temperature, Setpoint, P, I, and D to the Configured to Log box.



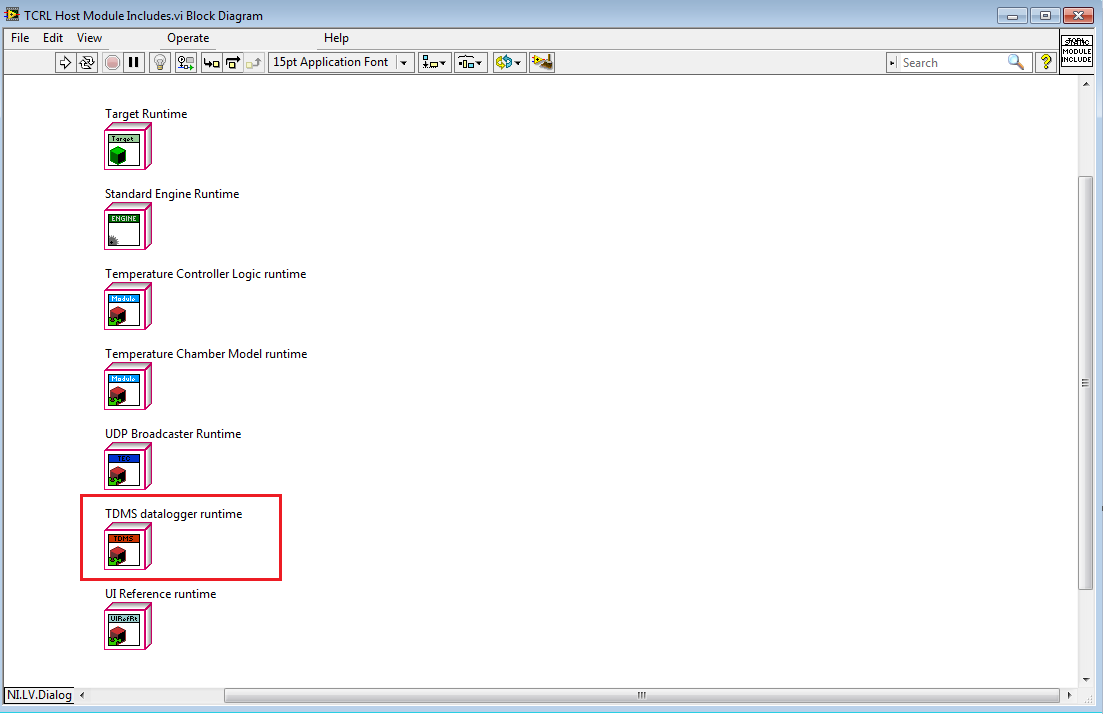
*Figure 2.2*

1. Go to the ***Datalogger Configuration*** tab. Press the first browse button to select a File path. Browse to [**\\Desktop\DCAF Hands On\Exercise 1**](file:///\\Desktop\DCAF%20Hands%20On\Temperature%20Controller\Exercise%201)**\Temperature Controller\Runtime** and create a folder named ***Data***. Type ***Temperature Measurements*** as the ***File name***.
2. Press the second browse button to select a Historical Directory. Browse to [**\\Desktop\DCAF Hands On\Exercise 1**](file:///\\Desktop\DCAF%20Hands%20On\Temperature%20Controller\Exercise%201)**\Temperature Controller\Runtime** and create a folder named ***Historical***. Type ***Temperature Measurements*** as the ***File name***. Verify your file paths with Figure 2.3.



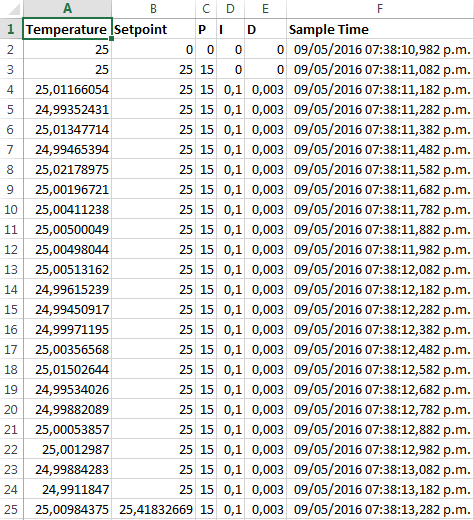
*Figure 2.3*

1. Since we added a new module, loaded classes should be updated. Use the scripting tool explained in ***Exercise 1>>Part 2>>Step 7*** to update the classes in **TCRL Host Module Includes.vi**. Verify the class has been successfully added to **TCRL Host Module Includes.vi.**



*Figure 2.4*

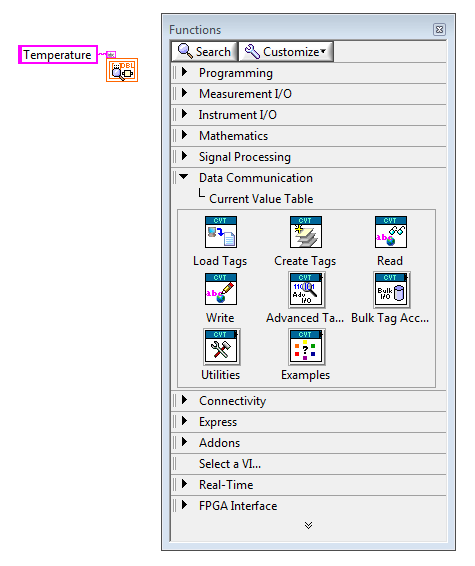
1. Run TCRL Host Main.vi. Do some changes to the setpoint and verify it still working and stop the VI.
2. Go to the Historical folder you created located at [**\\Desktop\DCAF Hands On\Exercise 1**](file:///\\Desktop\DCAF%20Hands%20On\Temperature%20Controller\Exercise%201)**\Temperature Controller\Runtime** and open the TDMS file just created. Verify the tags you added in the TDMS datalogger modules appear in the file and generated data.



*Figure 2.5*

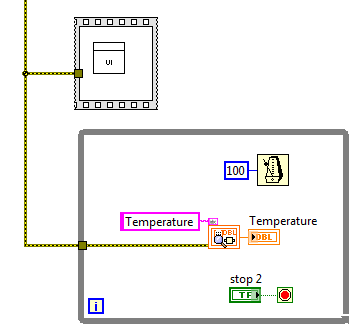
**Part 2: Add CVT**

1. Open the ***Temperature Controller*** project you developed in Exercise 1 if not already opened.
2. Open the ***Configuration Editor*** and load ***SimulatedSystem.pcfg*** if not already opened.
3. Add a CVT configuration module to the Simulation Engine in the same way you added the TDMS datalogger.
4. Select the CVT configuration module. Select To CVT direction. Move Temperature to the To CVT box. Save your configuration.
5. Open ***TCRL Host Main.vi*** Block Diagram. Add a ***Read*** VI from the ***Current Value Table Function Palette***. By default it is a double. Connect a string constant to the ***Tag Name Terminal*** and type ***Temperature***.



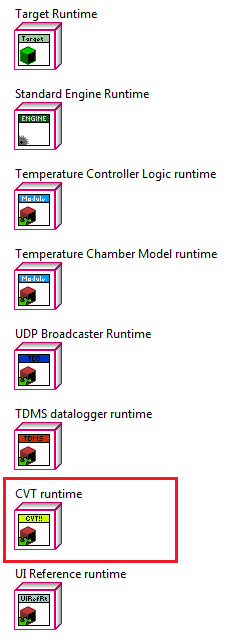
*Figure 2.6*

1. Finish the code as shown in Figure 2.7



*Figure 2.6*

1. Save the changes in TCRL Host Main.vi. Go back to the Configuration Editor and update the classes for TCRL Host Module Includes.vi as you did for the TDMS Datalogger Module. Verify the CVT class is added to TCRL Host Module Includes.vi.



*Figure 2.7*

1. Rearrange the Front Panel Indicators and run TCRL Host Main.vi. The UI should still be working. Verify the values in the UI correspond to the Temperature indicator in TCRL Host Main.vi

***End of Exercise 2***

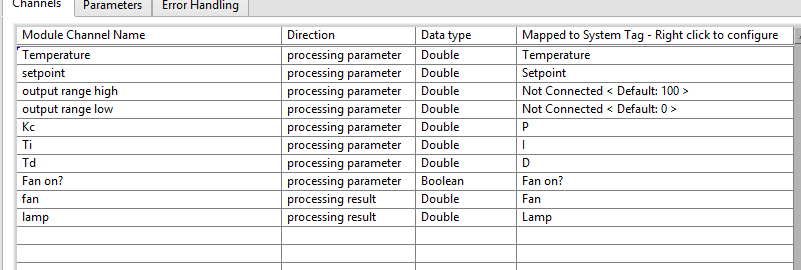
# Exercise 3: Creating a Custom Module

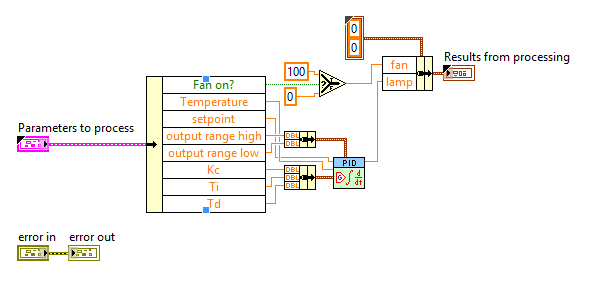
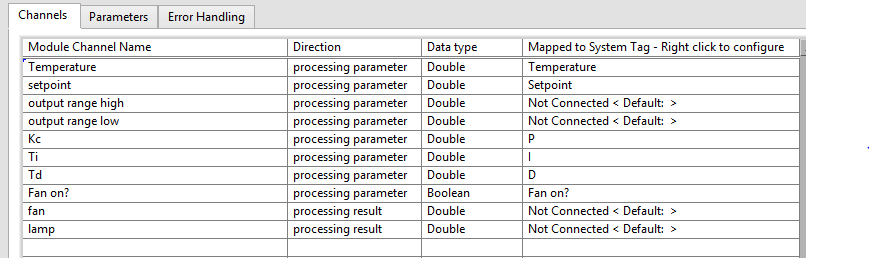
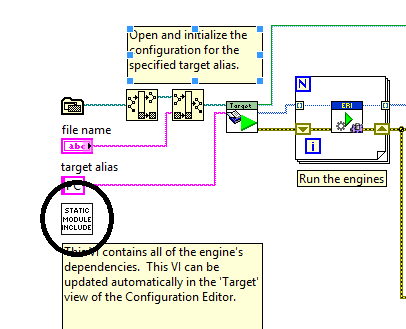
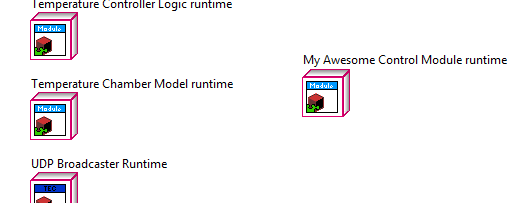
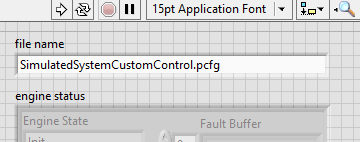
In this exercise we will implement a custom control algorithm for our temperature controller. This can be actually custom, or a simple PID controller as is used by the original example.

1. Navigate to **Project >> Create Project…**
2. In the tree on the left, select **TBDF >> Modules**.
3. Select **User Control Module**.
4. Enter a new name for your module and select an appropriate project path.
5. Add the following parameters to your new module as Channels. Channels represent data passed to or from your module during different execution stages, and channel names are case sensitive. The direction specifies whether the data is to the module or from the module, and is divided into inputs (data provided by input.vi), outputs (data provided to output.vi), and processing parameters and results (data passed to and from process.vi). For this hands on we will implement a processing step.

|  |  |  |
| --- | --- | --- |
| Name | Type | Direction |
| Temperature | Double | Processing parameter |
| setpoint | Double | Processing parameter |
| output range high | Double | Processing parameter |
| output range low | Double | Processing parameter |
| Kc | Double | Processing parameter |
| Ti | Double | Processing parameter |
| Td | Double | Processing parameter |
| Fan on? | Boolean | Processing parameter |
| fan | Double | Processing result |
| lamp | Double | Processing result |

At the end of this process you should have a channel configuration which looks like this:

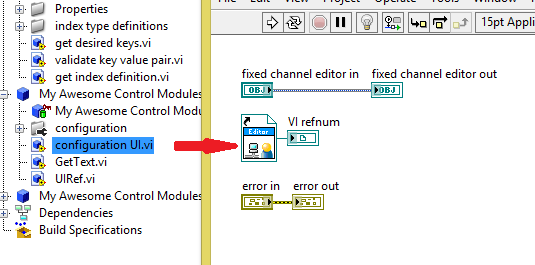
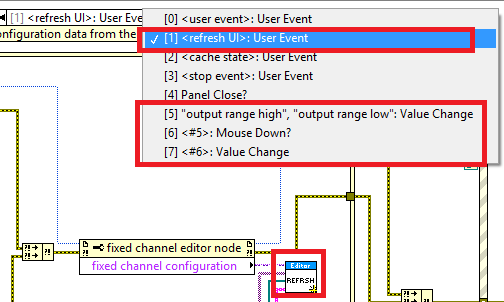


1. Press **Finish**.
2. Your new project will appear. Navigate to *YourModuleName* Runtime.lvclass and open *user process.vi*. This method should have two clusters, one input and one output, which match the list of tags above.
3. Open *process.vi* in the overrides folder. On either side, scripted accessors convert tag bus data into your user-defined cluster. These methods are *not* automatically generated by the project scripting tool and must be generated when the project is first scripted or after any change to the interface. If you need to make changes to the interface, go to Appendix A.  
   Note: During this process, the lvclass files are modified and so LabVIEW requires that your classes are not loaded in multiple contexts. That is, you must close any other projects which currently have the class loaded. If you see a “lock” icon over the class, the script will not work. To script these methods:
   1. Open **Tools >> TBDF >> Launch Control Module Scripting Utility…**
   2. Drag the runtime class from the project over the runtime class path control or browse for it manually, then do the same thing for the configuration class (*YourModuleName* Configuration.lvclass).
   3. Because you used the script, you can leave most options as the default and press **Run**.
4. Return to *process.vi*. *User process.vi* is the function being called in the middle and is the function you will implement. Add any control code you desire to this module. If you want to match the original module you just used, you can add the following:
   1. Drop down an instance of PID.vi from **Control and Simulation >> PID >> PID.vi**.
   2. Unbundle *Parameters to process.Temperature* and wire as the process value
   3. Wire *Parameters to process.setpoint* as the setpoint
   4. Bundle *Parameters to process.output range high and Parameters* to *process.output range low* and wire the cluster to output range
   5. Bundle *Parameters to process.Kc, Ti, and Td* and wire the cluster to PID gains
   6. Wire *Parameters to process.Fan on?* to a select function or case structure and set the two values as T=100 and F=0. Wire the output of this function to the *Results from processing.fan* value
   7. Wire the PID output to *Results from processing.lamp*.
   8. The result should look something like this:  
      
5. Save the new project and close it.
6. Reopen *Temperature Controller Example.lvproj* if you closed it, and reload the standard configuration editor (Open **Tools >> TBDF >> Launch Control Module Scripting Utility…**).
7. Navigate to **Tools >> Edit Plugin Search Paths**.
8. Press **Add** and navigate to the location of your new control module.
9. Reopen <LabVIEW 2015>\examples\TBD Examples\RT Temperature Controller\SimulatedSystem.pcfg and then to go **File >> Save As…** to make a copy of the configuration. For simplicity, save it in the same location but call it SimulatedSystemCustomControl.pcfg.
10. Now, **right click** on *Standard Engine (Simulation)* and select **Add >> *YourControllerModule***. Then, select this new module from the tree.
11. For each processing parameter tag, right click on the column “Mapped to System Tag” and configure the channel to be mapped to the appropriate system tag. You can look at “Temperature Controller Logic” to identify the correct mapping, which looks like this:  
    
12. Now, right click on the output ranges and press **Set** **Default** for each. Set output range high to 100 and output range low to 0 (this is not required, as the default if not set is 0). These are unmapped parameters, meaning they can be modified in the editor but they are constants at runtime – output range high will always be 100.  
    
13. Finally, we need to map our results to system tags. However, the editor prohibits mapping multiple writer channels (processing result or output) to a single system tag – mapping two writers to a single tag would introduce a race condition. So the next step is to either remove the mappings from the Temperature Controller Logic module, or to delete the module entirely. In either case the write reservations will be released. Reader channels may be ignored, as any tag can be read from multiple channels.
    1. Select Temperature Controller Logic, right click on rightmost column for the “fan” and “lamp” channels, and select **disconnect**.
    2. **And/Or:** Right click on Temperature Controller Logic in the tree and select **Remove.**
14. Return to your module’s configuration screen and map channel “fan” to tag “Fan” and channel “lamp” to tag “Lamp”.
15. Select to **File >> Save.**
16. Open Host Main.vi (our top-level application VI) and navigate to Host Module Includes.vi on the diagram. This function ensures that all appropriate modules are loaded into memory. You can also load precompiled modules (llb or lvlibp files) from disk, but for our purposes we will simply hardcode the appropriate modules.  
    
17. Drag an instance of *YourModuleName* Runtime.lvclass from the project onto the diagram of Host Module Includes.vi. This ensures that your new module is always loaded into memory.  
    
18. Now save and close this VI and return to Host Main.vi. On the front panel, enter the name of your configuration file (we suggested SimulatedSystemCustomControl.pcfg) in the *file name* control.  
    
19. Press **run**.
20. The behavior should match that of the original controller if you used the same code.   
    Debugging Note: While *process.vi* is a shared reentrant method and is therefore difficult to debug, *user process.vi* defaults to being non-reentrant and debuggable.

***End of Exercise 3***

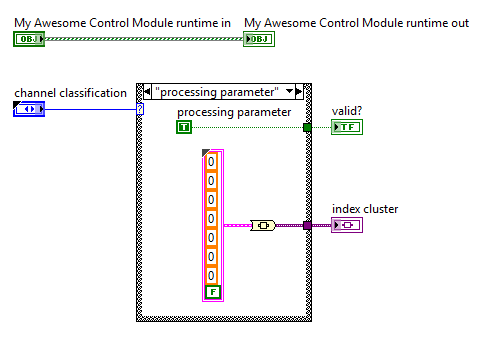
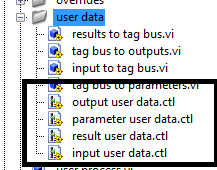
# Exercise 4: Creating a Custom Configuration Editor UI (Optional)

In this exercise, you will implement a custom user interface for configuring your controller module. This will likely take more time than we have available.

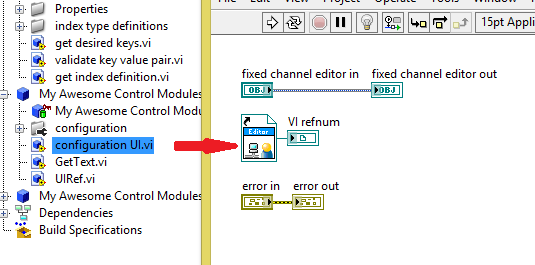
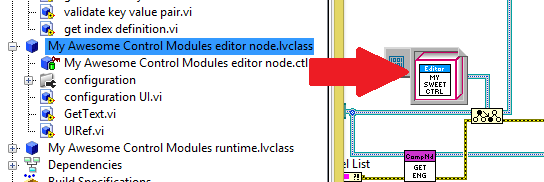
1. Close all projects except for the project you just scripted for your module.
2. To add a brand new editor you could follow the steps in Appendix A. This is not required, as we have provided a starting point.
3. To use the provided starting point, drag the contents of *<Desktop>/lesson three starting point* into *YourModuleName* editor node.lvclass.
4. Define *HandsOnCfgUI.vi* as your module’s editor.
   1. Right click on your editor node class and select **New >> VI for Override…**
   2. Select UI ref.vi and press **OK**.
   3. Remove the call parent method node, drop down a new **Static VI Reference** and drop your new *HandsOnCfgUI.vi* method into that static reference.  
      
5. Open *HandsOnCfgUI.vi* and look at the front panel. It is expected that this VI is broken right now. The controls provided are named identically to the channels in your module. This is for convenience. As you saw in the original, more complex UI, it is definitely possible to make the front panel more dynamic. Because time is limited, we’ve made the assumption that the editor node must be updated to match the channels in the configuration.
6. Fundamental to this particular implementation is that the combo boxes on the front panel map directly to channels in your module. That is, the combo box with the label “Temperature” represents data associated with the channel “Temperature”. In this case the data being displayed is what tag is mapped to that channel.
7. Go to the block diagram and look around. You will be implementing the code for the other VIs provided, which is called in the cases below. Each block diagram has instructions for what is required along with a list of all methods needed. These methods are all available on the palette, but we’ve picked out the appropriate ones to save you time. You’ll also see that much of the infrastructure (registering for the right events, locking and unlocking DVRs, etc) has been completed for you. The focus is just on using the TBDF APIs.   
   
8. If you run into any issues, ask for help or take a look at the solution.
9. Once you’ve completed making your changes, load up the standard configuration editor and try to use your new UI. Assuming you completed step 4 correctly, it should at least load correctly.   
   Debug Note: Once loaded, you can right click on the front panel (as shown in the subpanel of the main editor) and select “Open Block Diagram”). This makes it easy to debug your code as its running. If you need to make changes, you don’t need to stop the entire editor, you can simply click away from your module and this will unreserved your UI, allowing you to make edits.

***End of Exercise 4***

# Appendix A: Modifying control module data

1. If you made a mistake on your module or would simply like to add or remove different methods (for example, you want to add input functionality to your module), you will need to navigate to *get data definition.vi* in your runtime class. The “valid?” Boolean for each case of the case structure indicates whether or not the data interface is valid and if that method should be run. Setting a given Boolean to true tells the framework that you’d like to configure and run that method. If you followed the steps above, the Boolean should be true for *processing parameters* and *processing results*. The script will use these Booleans to determine if a given cluster is valid (the “placeholder” fields are just there to ensure the clusters are not broken).  
   
2. To add additional input or output channels, simply drop the appropriate controls or indicators into one of the four clusters in the *user data* virtual folder.  
   

# Appendix B: Creating a new editor UI

1. Right click on *YourModuleName* editor node.lvclass and select **Go To Parent Class**.
2. Open *configuration UI.vi* in the parent class (*control module editor.lvclass*) and save a copy into your editor class (save a copy into the project, then drag and drop into your editor). Be sure to name it something else, like *MyCfgUI.vi.*
3. Right click on your editor node class and select **New >> VI for Override…**
4. Select UI ref.vi and press **OK**.
5. Remove the call parent method node, drop down a new **Static VI Reference** and drop your new *MyCfgUI.vi* method into that static reference.  
   
6. Open your instance of *MyCfgUI.vi*, press **ctrl+m** and then **unlock**. Then, drag your editor class into the data value reference. This makes sure your new UI references your class rather than the parent class.  
   
7. The front panel shown is exactly what you have been working with so far. The event structure and surrounding code already have everything needed to cooperate with the framework. Specifically, be sure that you don’t remove the following event cases:
   1. [0] <user event>: User Event
   2. [1] <refresh UI>: User Event
   3. [2] <cache state>: User Event
   4. [3] <stop event>: User Event
   5. [4] Panel Close?