

# Hands-On 2: Static Modules

---

This hands-on covers the steps to create a DCAF Static module, these modules are easier to create and the recommended option if the module developer knows the number of channels that the module will require.

It is recommended that the Hands-On 1 is completed before starting with this one.

- [Hands-On 2: Static Modules](#)
  - [Introduction:](#)
  - [Exercise 1:](#)
    - [Part A: Create a Static PID Module](#)
    - [Part B: Modify the Initialization and Implement the User Process](#)
    - [Part C: Update the Static PID Module with Scripting](#)
    - [Part D: Running and Configuring the Module](#)
  - [Exercise 2:](#)
    - [Part A: Create the state machine module](#)
    - [Part B: Create a DCAF Application that runs the Static Module](#)

## Introduction:

---

DCAF Static modules are simple custom modules that are easy to create with the restriction that the module developer needs to know the number of channels that the module will have. By having the known number of channels, DCAF contains useful scripting tools that simplifies the module development.

Before creating a DCAF Static module there is an important checklist to cover:

- Is there a module that already covers this functionality?
- Is this a static or a dynamic module?
- What inputs or outputs my module needs?

In each exercise will go through this checklist during the module definition, and a more on how to define DCAF modules can be found in the DCAF developer guide in section 4.

## Exercise 1:

---

In this exercise we will create a DCAF Static module that contains a custom PID control for our application.

### Concepts Covered:

- Static Module Creation
- Basic Static Module overrides
- Static channel Configuration
- Working with the User Process VI
- Static Module Scripting

### Module Definition:

This is processing module that will contain a single PID. Initially it will have some values hardcoded and will be updated to change so they can be updated by the engine.

- Is there a module that already covers this functionality?  
In this case yes there is a PID module for DCAF that could be used, but as we want to go with the exercise we are going to ignore it.
- Is this a static or dynamic modules?  
The module only needs 1 pid, and we know the inputs required for the PID so it is a static module.

-What inputs or outputs my module needs?

This is a processing module all its inputs come from the Tag Bus and all the outputs go into the Tag Bus. The initial channel list is the following one:

Name	Type	Direction
Temperature	Double	Processing Parameter
Setpoint	Double	Processing Parameter
Output	Double	Processing Result

This channel list will change during the exercise.

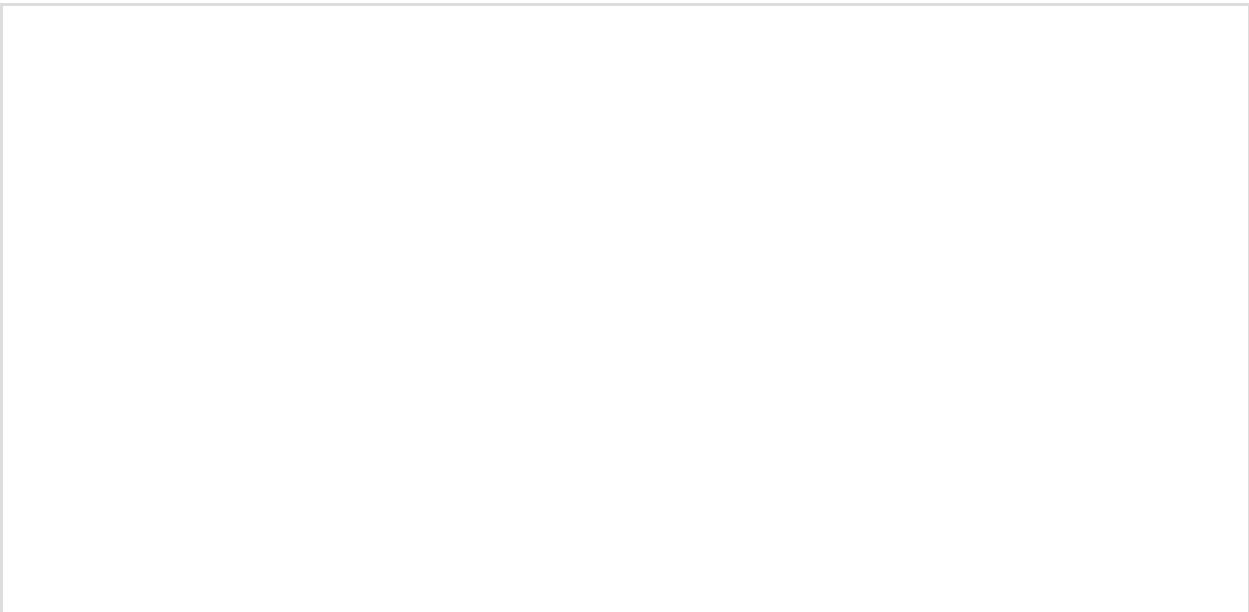
## Part A: Create a Static PID Module

### Create module from template

1. Navigate to **Project >> Create Project...**
2. In the tree on the left, select **DCAF >> Modules** and select **DCAF Static Channel Module**.
3. Type in **Custom Temperature Controller** as the **Module Name**.
4. Look for the Project Root **\\Desktop\\Hands On\\Hands-On\_2\\Exercises\\Temperature Controller\\Modules** and create a folder called **Custom Temperature Controller**. Get into that folder and press **Current Folder**.
5. Add the following parameters to your new module as **Channels** in the **Channel Specifications** tab. 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	Double	Processing Result

Your configuration should look like the one shown in Figure 1.1.



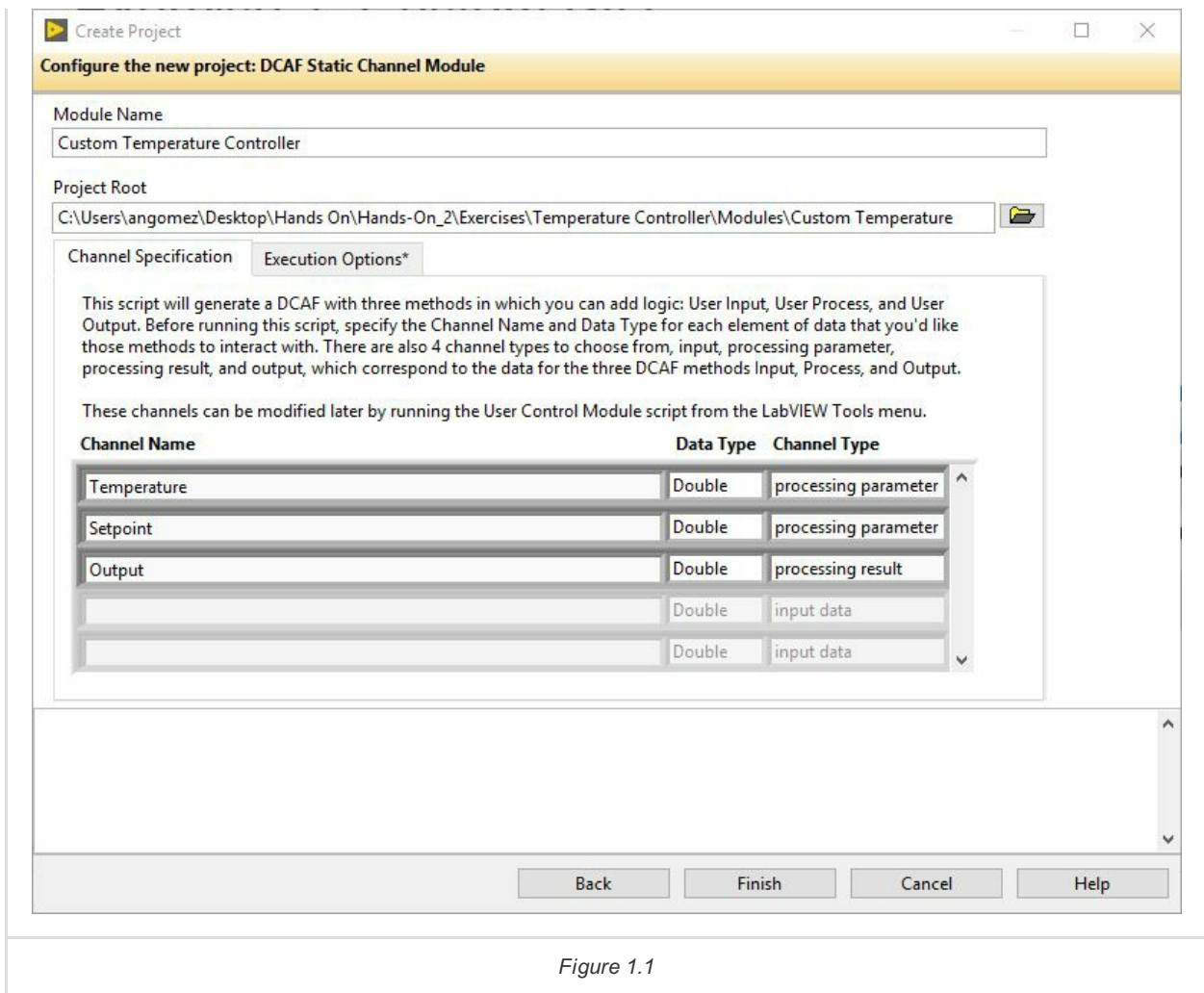
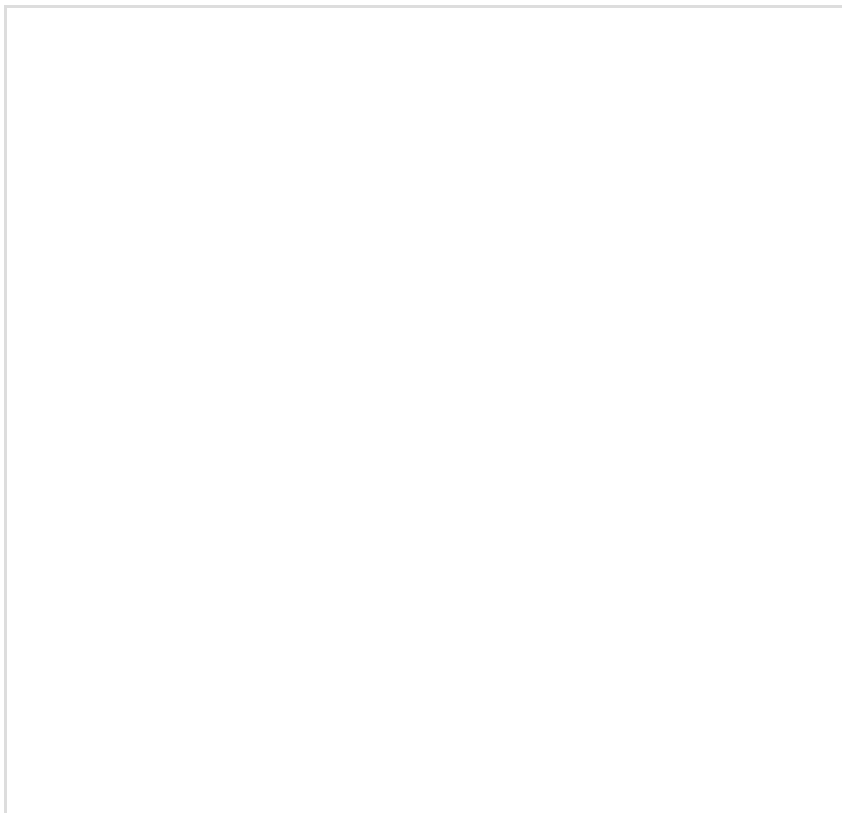


Figure 1.1

7. Press Finish. Wait for a few seconds for the full project to be scripted



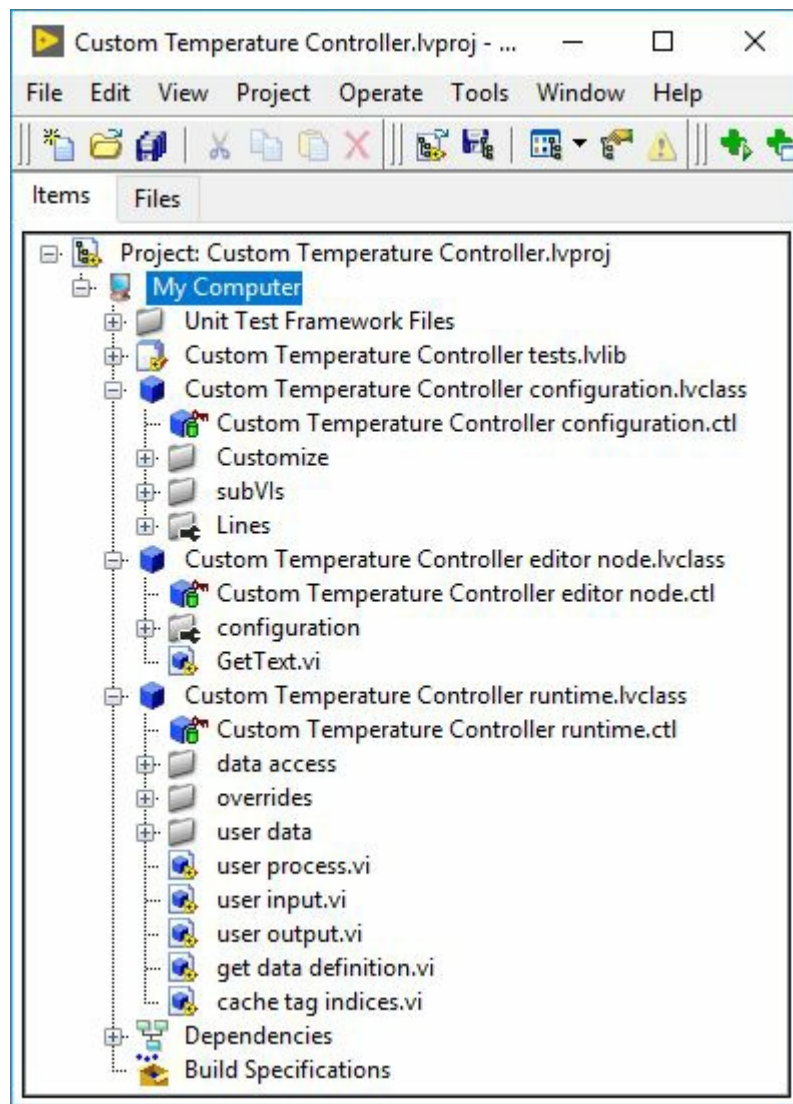
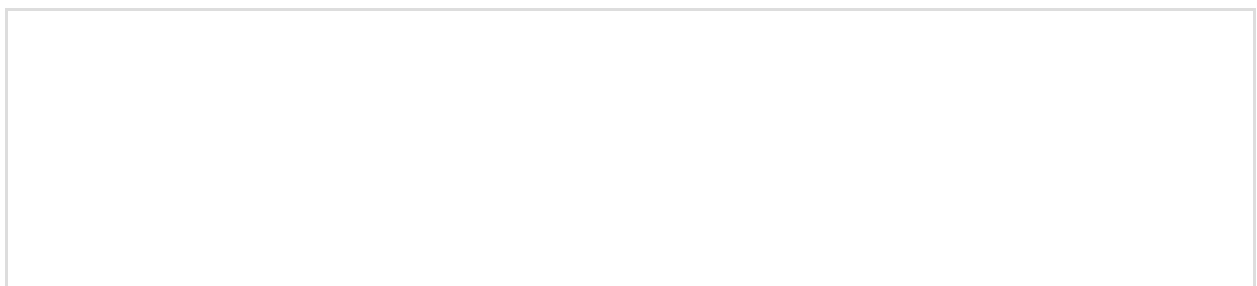


Figure 1.2

8. Your new project will appear and look similar to the image in figure 1.2. Feel free to close the accessor VIs that show up when the scripting is done. We will now explore how this new module loads from the editor. Open the DCAF editor by going to **Tools >> DCAF >> Launch Standard Configuration Editor**.
9. We want to configure the editor to search for modules in the path where our PID module was saved. From the DCAF Editor, navigate to **Tools >> Edit Plugin Search Paths....** Add the next path as a search path: **\\Temperature Controller\\Modules**. Click OK for the editor to refresh the list of plugins available.  
**Note:** Add the whole modules directory as the simulation module is in that folder.
10. Add a PC target and a standard engine under the system to finally add your Custom Temperature Controller Module under the engine. It should look as in the image below. As you can see, the channels of our module are ready to be connected to tags in the engine.



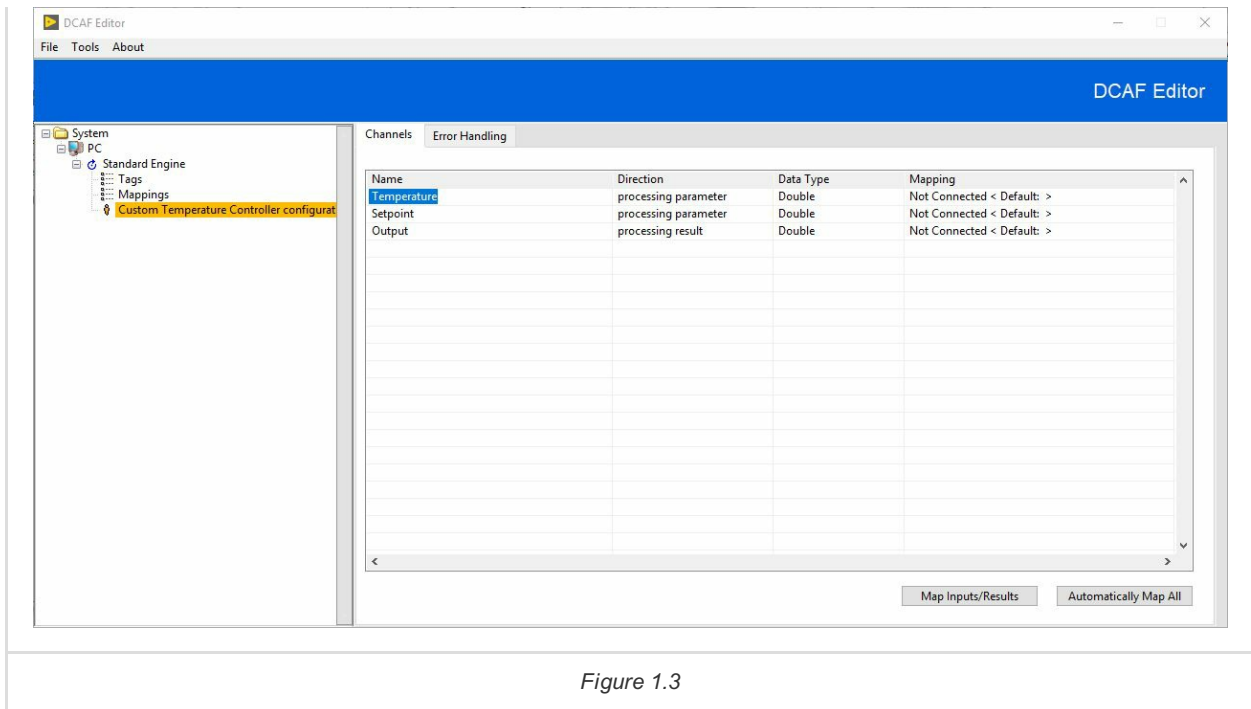


Figure 1.3

11. Close the DCAF Editor and for now don't save any changes.
12. We will now modify our module to contain parameters: Output Range High and Output Range Low. Find and open the **get desired keys.vi** in the configuration class. This should be located in the Customize folder.
13. Modify this VI to return the name of the parameters Output Range High and Output Range Low in the desired keys output. Close and save this VI.

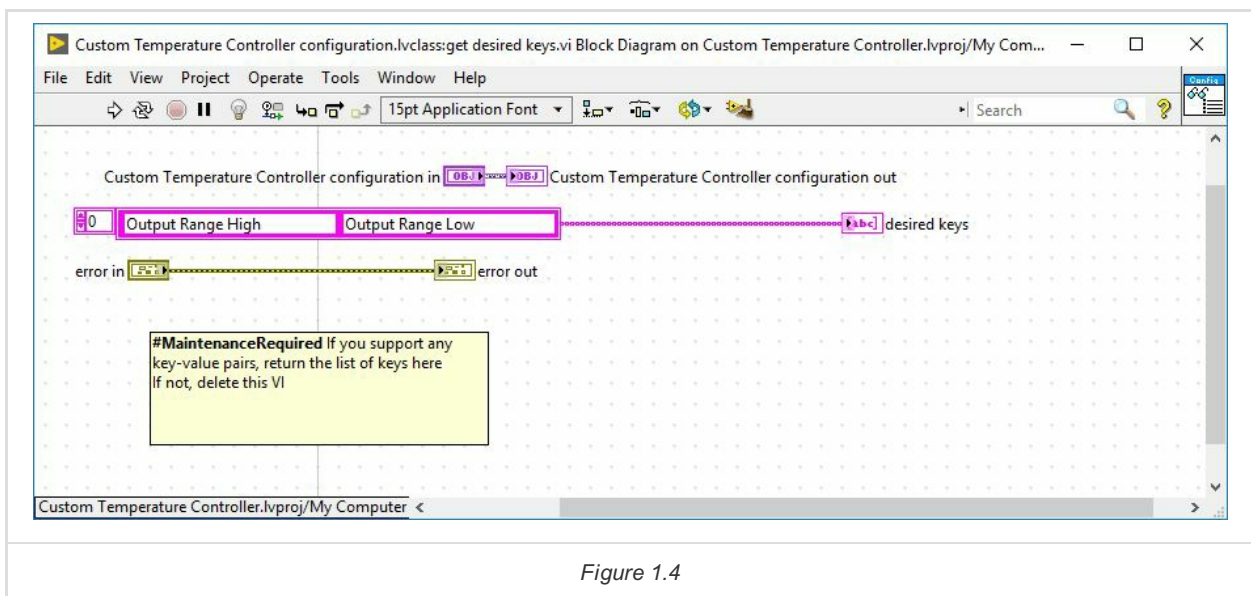
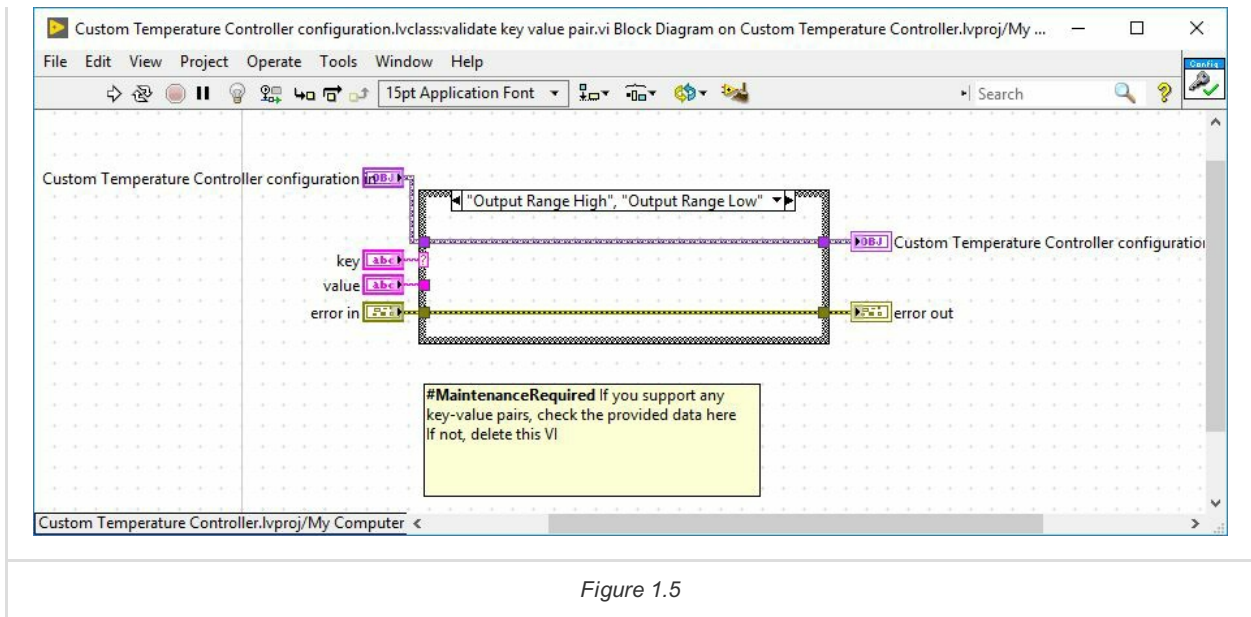
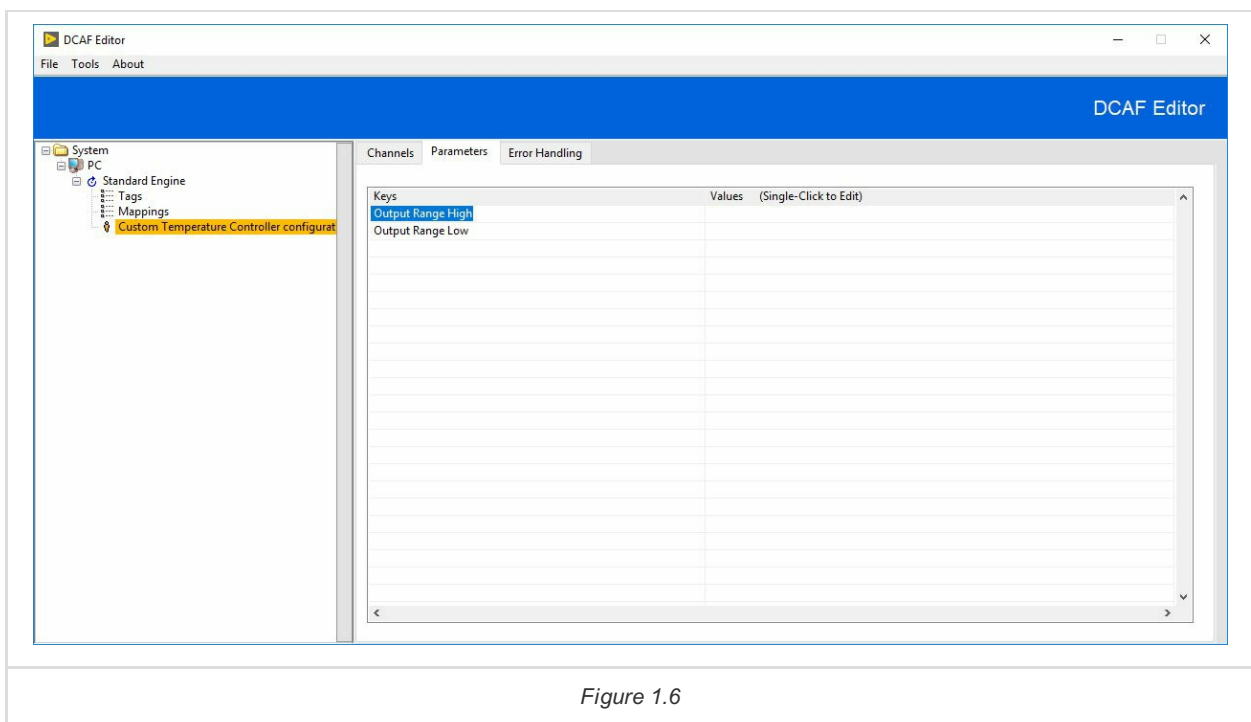


Figure 1.4

14. Find and open the **validate key value pair.vi** in the configuration class. This should be located in the Customize folder as well. This VI runs each time the user changes the value of our parameters. In this case we will just add cases to handle the keys Output Range High and Output Range Low without checking anything in particular about the value. Check the image below to see the expected result for this VI. Save it and close it when done.



- Open the DCAF Editor and add our module again. After our recent changes, a new tab called Parameters shows up where the user can configure the value for Output Range High and Output Range Low. Close the DCAF editor and don't save any changes for now.



We are done with building the configuration code for our static module. For static modules, most of the work is scripted by us from the template. In the next section of this exercise we will mainly work on the runtime class of the module.

## Part B: Modify the Initialization and Implement the User Process

We are now ready to perform the necessary implementations in the runtime side of our module. We will start by modifying the initialization to be able to load the parameters that will be used by our process. Once this is done, we will modify the user process to contain a PID.

- We will start by modifying our runtime class private data to contain the parameters we want to save. Open the **Custom Temperature Controller runtime.ctl** to contain a cluster with the Output Range High and Output Range Low. Make sure

you keep the rest of this private data unmodified. Consult figure 1.7 to see the expected result.

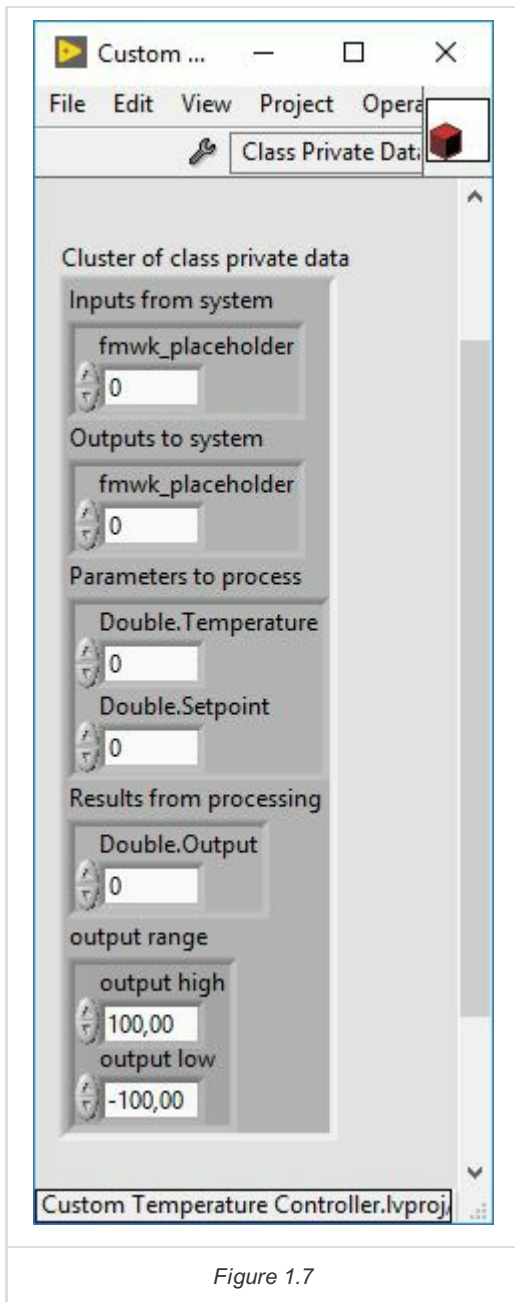
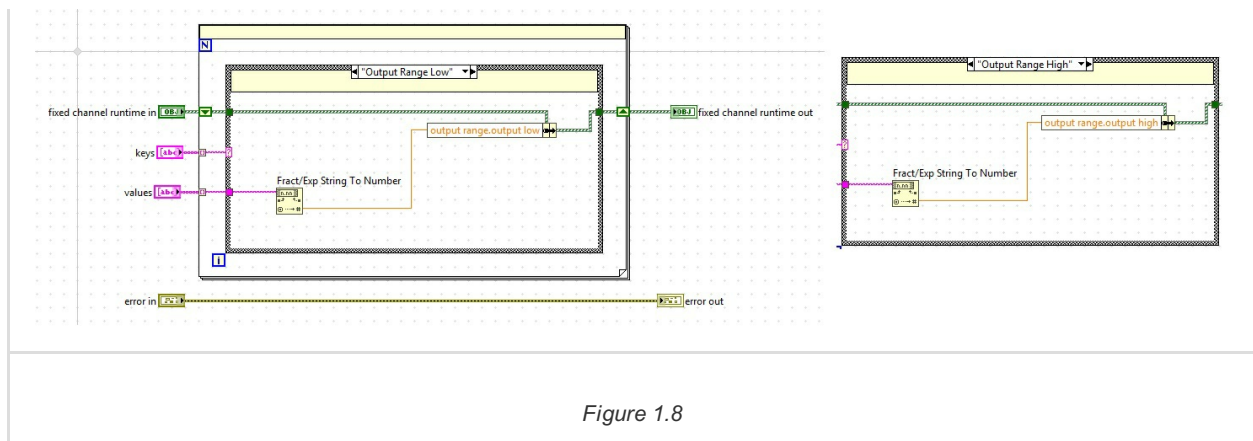


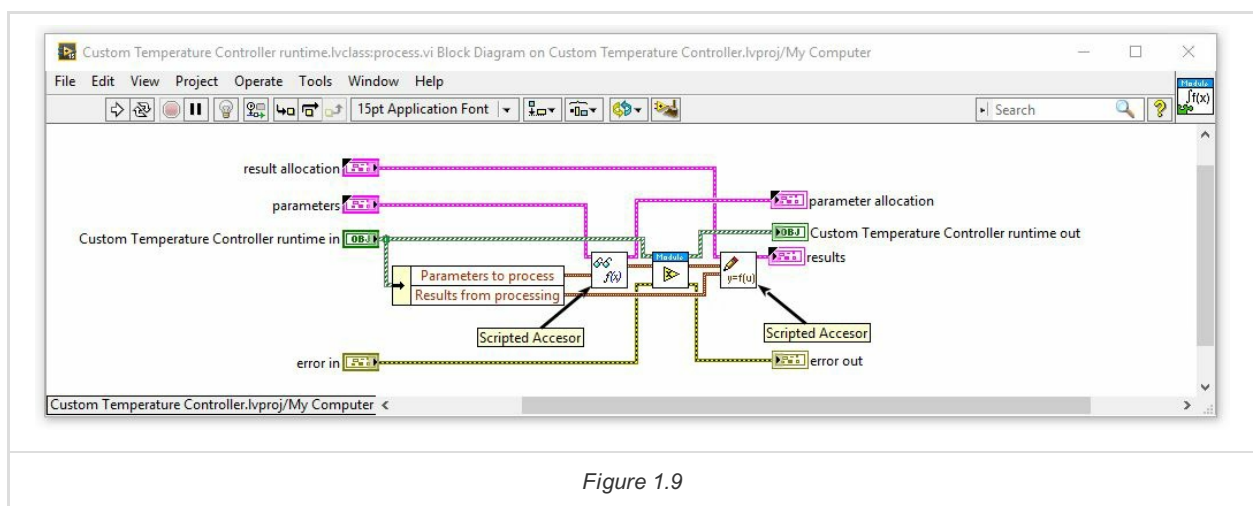
Figure 1.7

2. Next, create an override for the **apply key value pairs.vi** to store the parameters that were saved as keys into the **Custom Temperature Controller runtime.ctl**. To create the override VI, right-click on the runtime class and select **New >> VI for Override** and select the **apply key value pairs.vi** in the *New Override* window.
3. Modify the **apply key value pairs.vi** to look like the one in the image below. When the Output Range High or Output Range Low key is detected we store its value in the corresponding element in our private data. Save the modified VI in the **\Temperature Controller\Modules\Custom Temperature Controller\module\execution** folder.



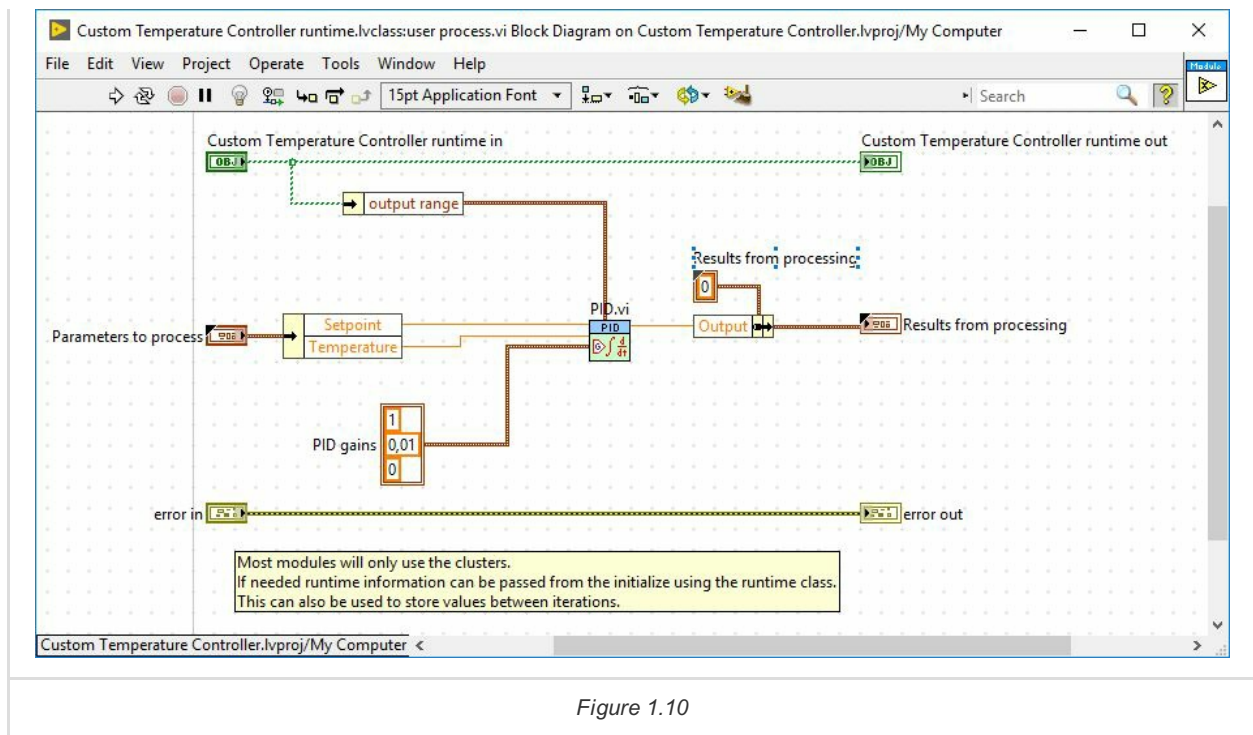
- a. Other than the two cases shown in the image, there is a \*Default\* case that simply passes runtime object from the input
- b. Make sure you use shift registers to store the object class in the for loop.
- c. Make sure the VI doesn't have a broken arrow when you finish modifying it.

4. Navigate to **Custom Controller Module runtime.lvclass** and 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 and must be generated when the project is first scripted or after any change to the interface. We will cover how to make changes to this VIs in Part C of this exercise. Feel free to explore with more detail the code inside these accessors.



5. Navigate to **Custom Controller Module runtime.lvclass** and open **user process.vi**. This method should have two clusters, one input and one output, which match the list of channels we configured for our module. Implement the same PID control as shown in the Figure 1.10.





1. Drop down an instance of **PID.vi** from **Control and Simulation >> PID >> PID.vi**.
2. Drop down the **Unbundle by Name** function.
3. Wire the **Unbundle by Name** function to the *Custom Temperature Controller runtime in* object. Access with this function the output range cluster with the high and low limits for the PID.
4. Wire **output range** to **PID.vi output range input**.
5. Wire **Parameters to process.Temperature** to **PID.vi process variable input**.
6. Wire **Parameters to process.setpoint** to **PID.vi setpoint input**.
7. Create a constant from **PID.vi PID gains input**.
8. Wire PID.vi output to **Output** from **processing.Output**.
9. The result should look something like Figure 1.10.

Our Custom Temperature Controller Static Module is now ready to run. However, we will study the steps to update our module with new processing parameters in the part C.

## Part C: Update the Static PID Module with Scripting

As with any project, requirements changed and now we need to add channels for the P, I and D values. In this section we are going to update our module channel list to match the table below :

Name	Type	Direction
Temperature	Double	Processing Parameter
Setpoint	Double	Processing Parameter
P	Double	Processing Parameter
I	Double	Processing Parameter
D	Double	Processing Parameter
Output	Double	Processing Result

**Update the clusters**

1. Open the **parameter user data.ctl**

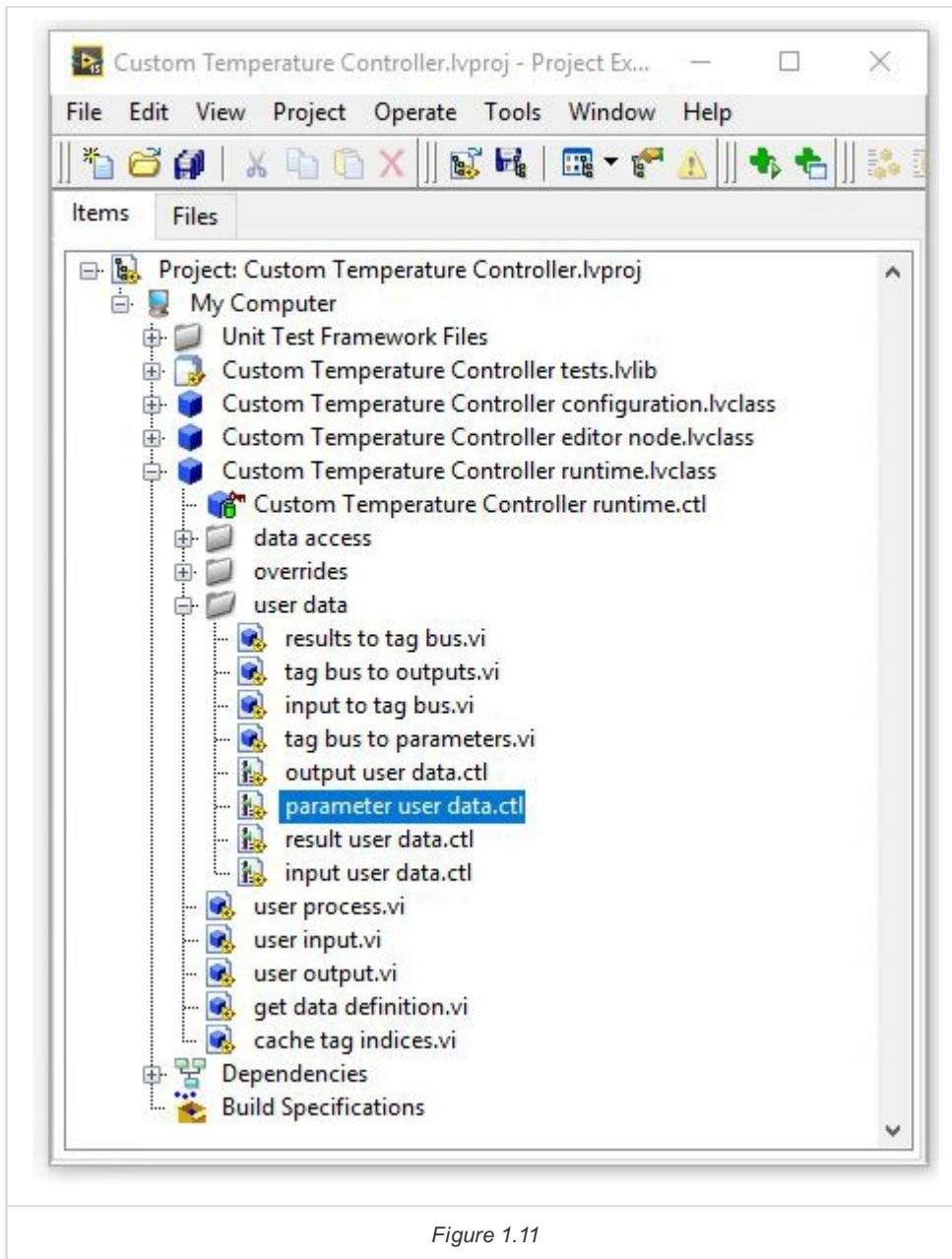
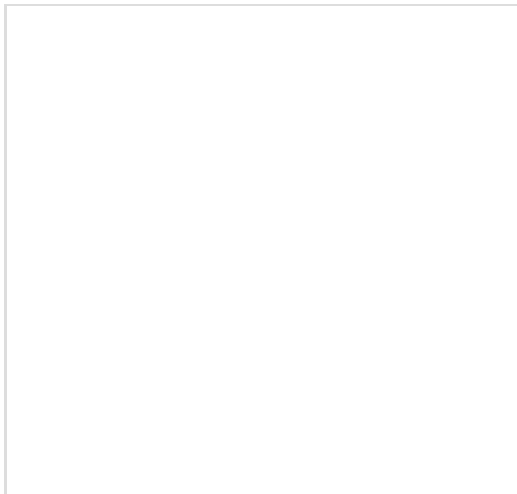


Figure 1.11

2. Add the new channels for the cluster to look like in figure 1.12



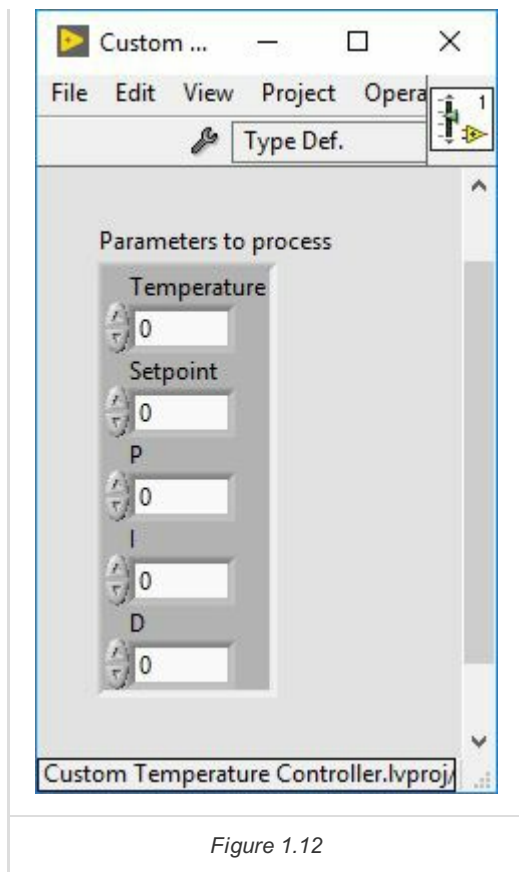


Figure 1.12

3. Save the control and apply changes **File>>Apply Changes**

**Note:** If you are adding channels of a different type that the module previously didn't have like a input channel in this case. It is necessary to modify **get data definition.vi** in the 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).

#### Run the scripting

**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.

4. Open **Tools >> DCAF >> Launch Control Module Scripting Utility...**
5. Drag the runtime class from the project over the runtime class path control or browse for it manually, then repeat for the configuration class (YourModuleName Configuration.lvclass).  
You can leave the rest of the options as the default and run this VI. Once the script finishes executing, the **Custom Temperature Controller runtime.lvclass:tag bus to parameters.vi** should look similar to the one in the figure 1.13 below.

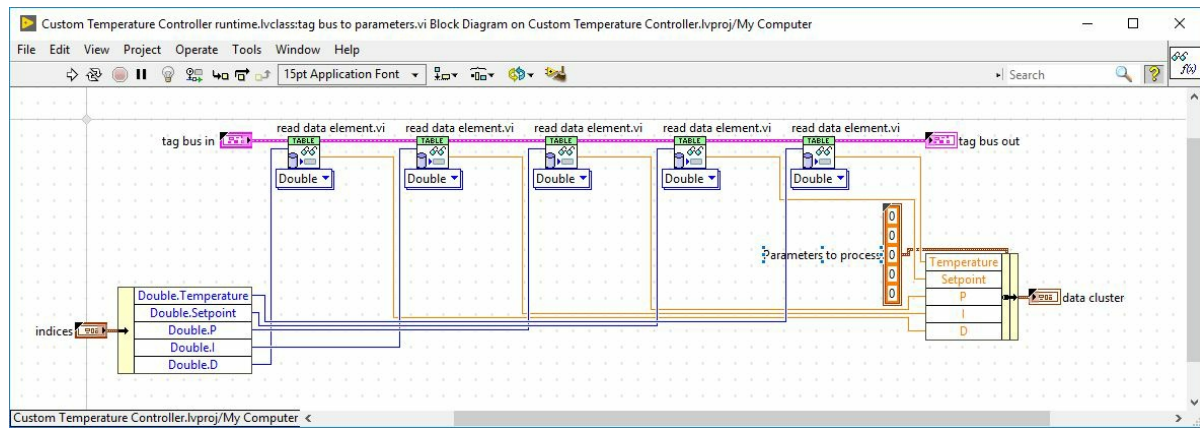


Figure 1.13

6. Navigate again to **Custom Controller Module runtime.lvclass**, open **user process.vi** and modify it to take the P,I and D gains from the parameters to process cluster. Your final VI should look similar to the one in figure 1.14.

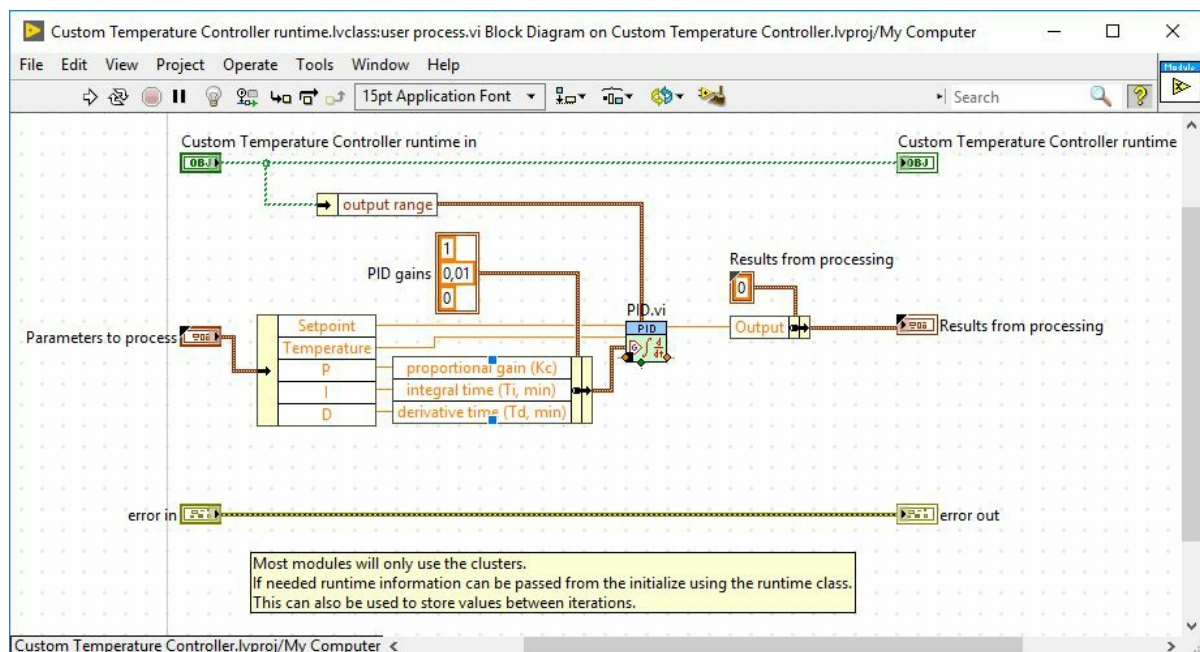


Figure 1.14

7. Save the VI and module project. Close the project since now we will use this module from a separate project.

## Part D: Running and Configuring the Module

### Configuration Editor

1. Open Standard Configuration Editor (**Open Tools >> DCAF >> Launch Standard Configuration Editor...**).
2. Open **Static Temperature Chamber.pfcg** in the **\Desktop\Hands On\Hands-On\_2\Exercises\Temperature Controller** folder.
3. Right click on Standard Engine and select **Add >> Other>>Custom Temperature Controller**. Then, select this new module from the tree.
4. For each processing parameter tag, click on the column *Mapping* 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 Figure 1.15:

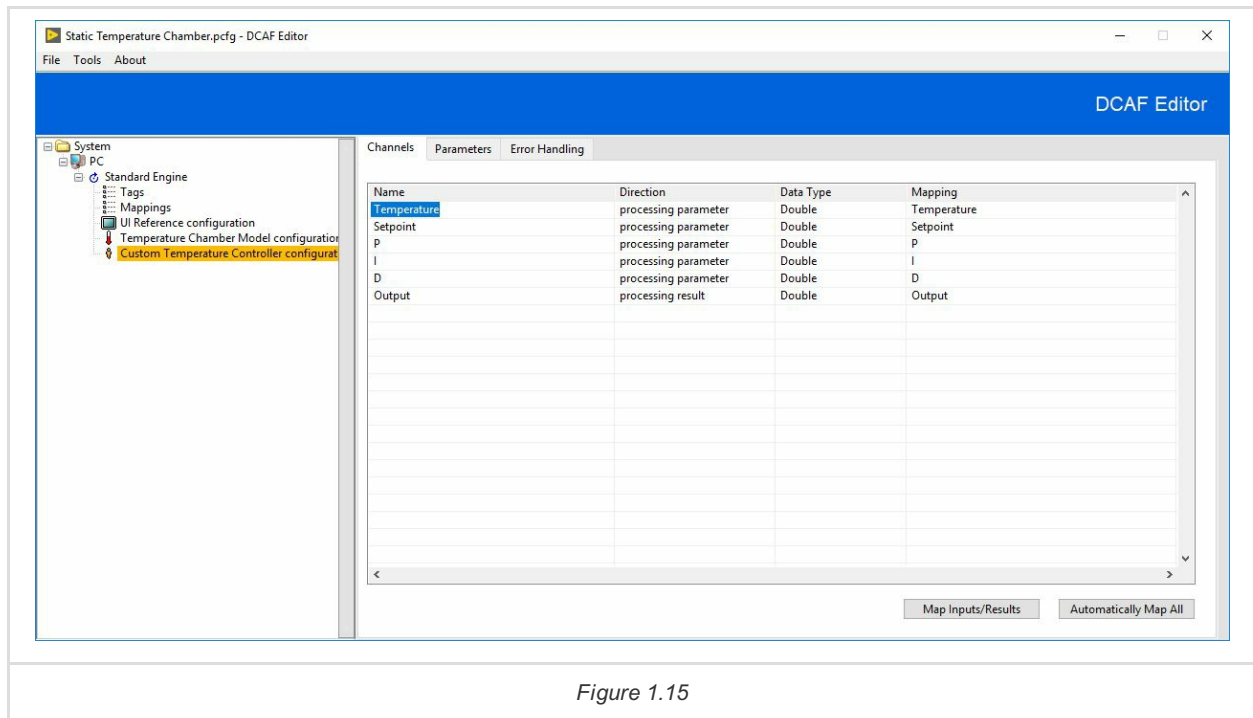


Figure 1.15

- Now, navigate to the Parameters tab and configure the value for the range limits. Set output range high to 100 and output range low to 0. These are not channels, meaning they can be modified in the editor but they are constants at runtime – output range high will always be 100.

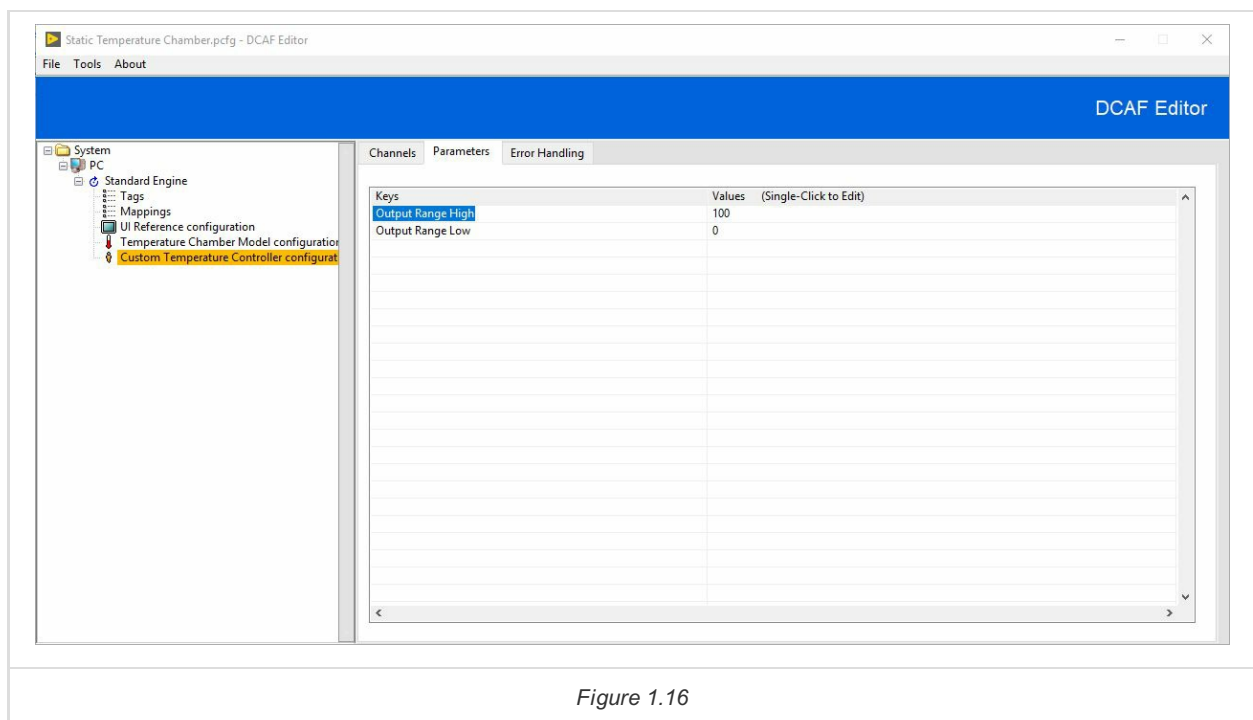
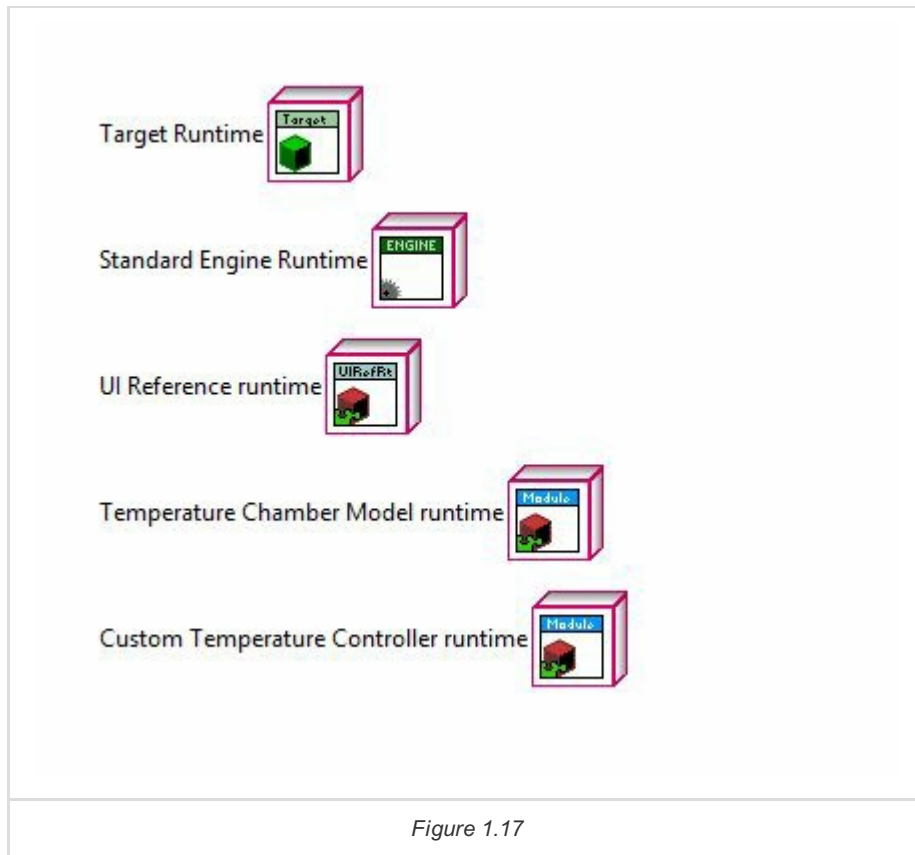


Figure 1.16

- Script the corresponding includes VI to contain all the dependencies classes for our configuration. In the DCAF tree, go to PC and configure the *Includes file path* to the **Temperature Chamber Includes.vi** in the **\\Desktop\\Hands On\\Hands-On\_2\\Exercises\\Temperature Controller** folder. Click on Generate to script this VI.
- Select to File >> Save.
- Open the **Temperature Chamber.lvproj** in **\\Desktop\\Hands On\\Hands-On\_2\\Exercises\\Temperature Controller**.

- From the Project Window, open **Temperature Chamber Includes.vi**. This function ensures that all appropriate modules are loaded into memory. You can also load precompiled modules (.llb or .lvlib files) from disk, but for our purposes we will simply hardcode the appropriate modules.



### Running the Code

- Open the **Temperature Chamber Main.vi**.
- Set the Configuration File Path to the path of the *Static Temperature Chamber.pcfg*.

**Note:** If the configuration file name is not valid you will receive error 538500

- Run the VI and use the UI to verify the PID behaves as expected.

## Exercise 2:

In this exercise we will create a DCAF Static module that contains a state machine. This is an optional and advanced exercise so there are no detailed instructions. To create this module you can use exercise 1 as reference and the DCAF developer guide section 4. There is a solution included in the solutions folder.

### Concepts Covered:

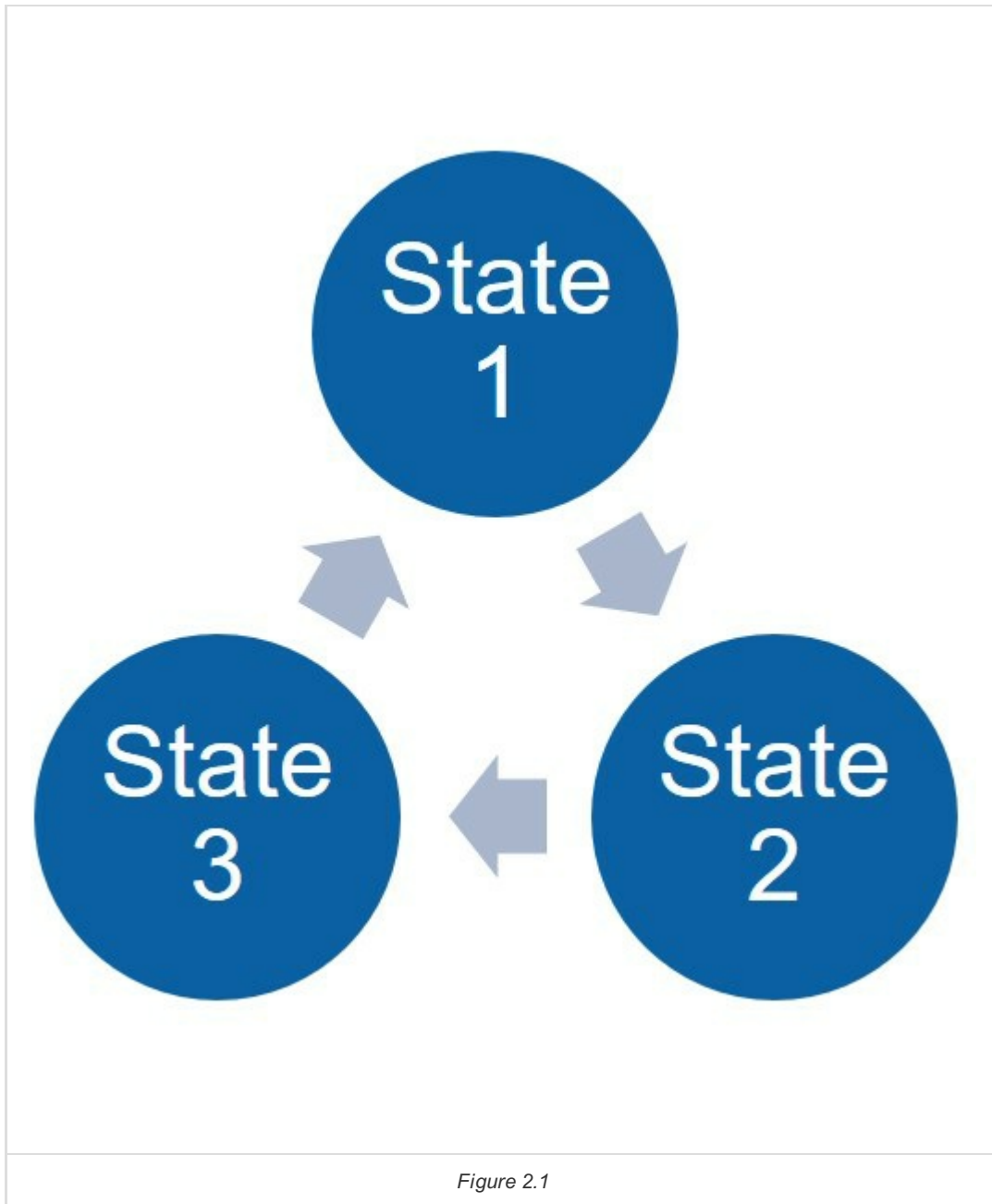
- Use of runtime class private data
- Module initialization
- Static module parameters

### Module Definition:

The module will be a simple state machine that contains 3 states. These states will transition from one state to the next only if the correct Boolean is selected.

The state machine will have simple transitions.

- Change from state 1 to state 2 if only A is True
- Change from state 2 to state 3 if only B is True
- Change from state 3 to state 1 if only C is True



The inputs and outputs list for this modules are:

Name	Type	Direction
A	Double	Processing Parameter
B	Double	Processing Parameter
C	Double	Processing Parameter
State 1	Double	Processing Result



State 2	Double	Processing Result
State 3	Double	Processing Result

In addition to this the module will have a parameter that defines the initial state:

-initial\_state: I32

## Part A: Create the state machine module

1. Create module from template
2. Add initial state parameter.
3. Modify initialization in runtime class.
4. Implement the user process
5. Test the user process stand alone

## Part B: Create a DCAF Application that runs the Static Module

1. Create a new project with the Basic Execution Template
2. Create a new vi for the UI that looks like figure 2.2
3. Create a DCAF configuration that maps the state machine to the UI using the UI modules
4. Run the application and test.

