Hands-On 2: Static Modules

This hands-on covers the creating of a DCAF Static module, this modules are easier to create and the recommended option if the module developer know s the number of channels that the module w ill require.
It is recommended that the Hands-On 1 is completed before doing this one.

Introduction:

DCAF Static modules are simple custom modules that are easy to create w ith the restriction that the module developer needs to know the number of channels that the module w ill have. By having the know n number of channels, DCAF contain 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 w ill 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 w e w ill 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 w ith the User Process VI
- Static Module Scripting

Module Definition:

This is processing module that w ill contain a single PID. Initially it w ill have some values hardcoded and w ill 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 w e w ant to go w ith the exercise w e are going to ignore it.
  -Is this a static or dynamic modules?
  The module only needs 1 pid, and w e 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 follow ing one:

Name	Type	Direction
Temperature	Double	Processing Parameter
Setpoint	Double	Processing Parameter
Output Range High	Double	Processing Parameter
Output Range Low	Double	Processing Parameter
Output	Double	Processing Result

This channel list w ill change during the exercise.

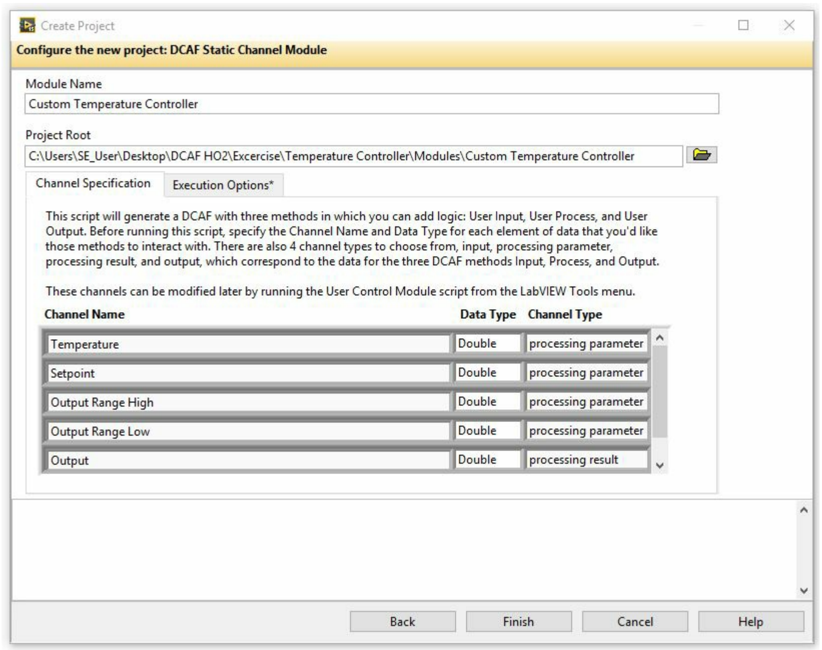
Part A: Create a Static PID Module

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 **Custom Temperature Controller** as the **Module Name**.
4. Look for the Project Root **\Temperature Controller\Modules** and create a folder called **Custom Temperature Controller**. Get into that folder and press **Current Folder**.
5. Add the follow ing 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 w hether 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 w e w ill 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
Output	Double	Processing Result

Your configuration should look like the one show n in Figure 1.1.



Create Project

Configure the new project: DCAF Static Channel Module

Module Name

Custom Temperature Controller

Project Root

C:\Users\SE_User\Desktop\DCAF HO2\Excercise\Temperature Controller\Modules\Custom Temperature Controller

Channel Specification | Execution Options*

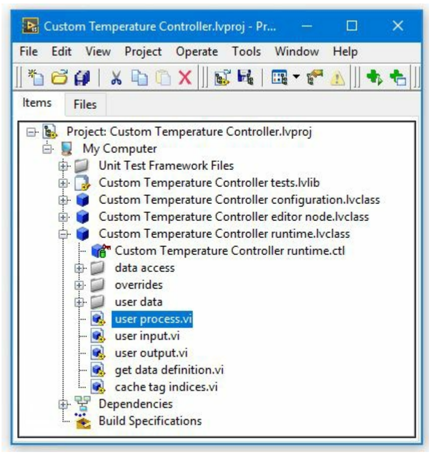
This script will generate a DCAF with three methods in which you can add logic: User Input, User Process, and User Output. Before running this script, specify the Channel Name and Data Type for each element of data that you'd like those methods to interact with. There are also 4 channel types to choose from, input, processing parameter, processing result, and output, which correspond to the data for the three DCAF methods Input, Process, and Output.

These channels can be modified later by running the User Control Module script from the LabVIEW Tools menu.

Channel Name	Data Type	Channel Type
Temperature	Double	processing parameter
Setpoint	Double	processing parameter
Output Range High	Double	processing parameter
Output Range Low	Double	processing parameter
Output	Double	processing result

Figure 1.1

7. Press Finish.

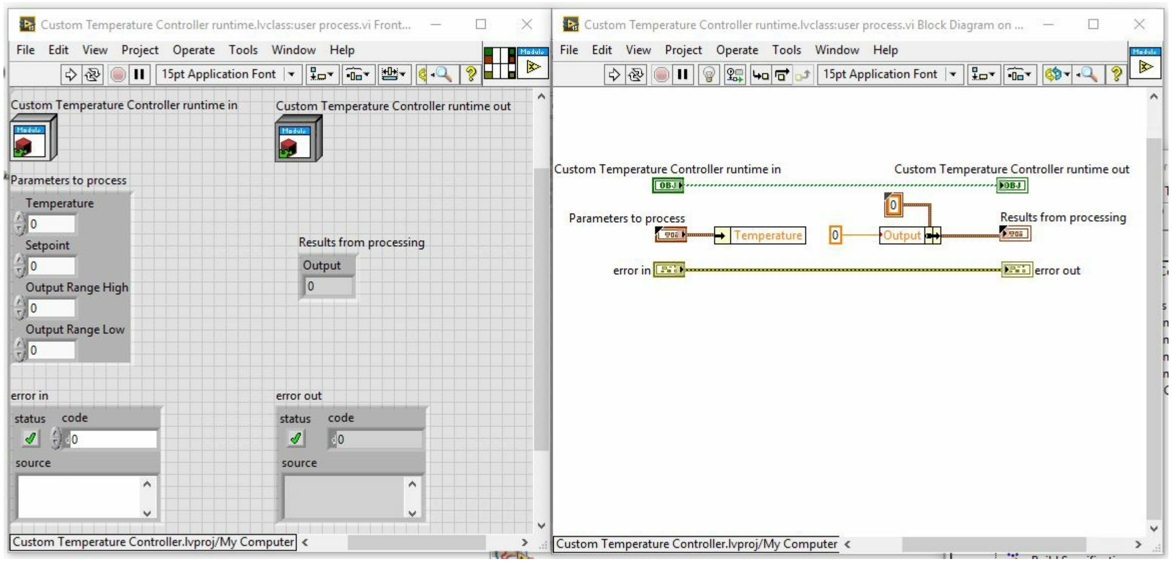


Custom Temperature Controller.lvproj - Pr...

```
Project: Custom Temperature Controller.lvproj
  My Computer
    Unit Test Framework Files
    Custom Temperature Controller tests.lvlib
    Custom Temperature Controller configuration.lvclass
    Custom Temperature Controller editor node.lvclass
    Custom Temperature Controller runtime.lvclass
      Custom Temperature Controller runtime.ctl
      data access
      overrides
      user data
      user process.vi
      user input.vi
      user output.vi
      get data definition.vi
      cache tag indices.vi
    Dependencies
    Build Specifications
```

Figure 1.2

8. Your new project w ill appear. Navigate to **Custom Controller Module runtime.lvclass** and open **user process.vi**. This method should have tw o clusters, one input and one output, w hich match the list of tags above. This clusters w ere created based on the table you just filled in.



Custom Temperature Controller runtime.lvclass:user process.vi Front...
Custom Temperature Controller runtime.lvclass:user process.vi Block Diagram on ...
File Edit View Project Operate Tools Window Help
15pt Application Font
Custom Temperature Controller runtime in
Custom Temperature Controller runtime out
Parameters to process
Temperature
0
Setpoint
0
Output Range High
0
Output Range Low
0
Results from processing
Output
0
error in
status code
0
source
error out
status code
0
source
OBJ
Temperature
0
Output
error in
error out
Custom Temperature Controller.lvproj/My Computer

Figure 1.3

9. Open process.vi in the overrides folder.

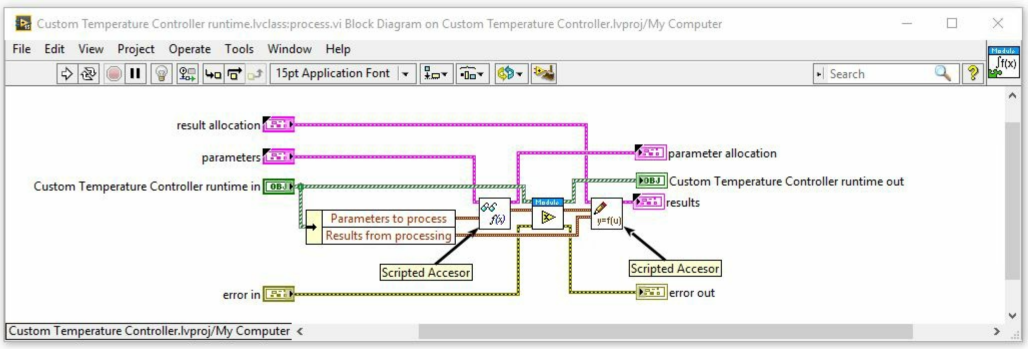

Custom Temperature Controller runtime.lvclass:process.vi Block Diagram on Custom Temperature Controller.lvproj/My Computer
File Edit View Project Operate Tools Window Help
15pt Application Font
Search
result allocation
parameters
Custom Temperature Controller runtime in
Parameters to process
Results from processing
Scripted Accesor
error in
parameter allocation
Custom Temperature Controller runtime out
results
Scripted Accesor
error out
Custom Temperature Controller.lvproj/My Computer

Figure 1.4

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 w hen the project is first scripted or after any change to the interface. w e w ill cover how to make changes to this VIs in Part B of this exercise. This accessors w ill look like figure 1.5.

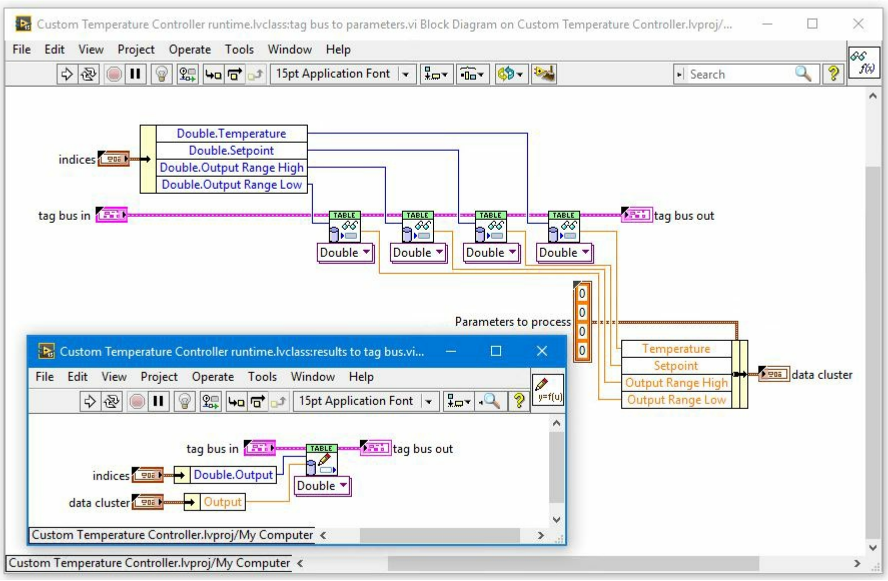

Custom Temperature Controller runtime.lvclass:tag bus to parameters.vi Block Diagram on Custom Temperature Controller.lvproj/...
File Edit View Project Operate Tools Window Help
15pt Application Font
Search
indices
Double.Temperature
Double.Setpoint
Double.Output Range High
Double.Output Range Low
tag bus in
Double
Double
Double
Double
tag bus out
Parameters to process
Temperature
Setpoint
Output Range High
Output Range Low
data cluster
Custom Temperature Controller runtime.lvclass:results to tag bus.vi...
tag bus in
tag bus out
indices
Double.Output
Double
data cluster
Output
Custom Temperature Controller.lvproj/My Computer
Custom Temperature Controller.lvproj/My Computer

Figure 1.5

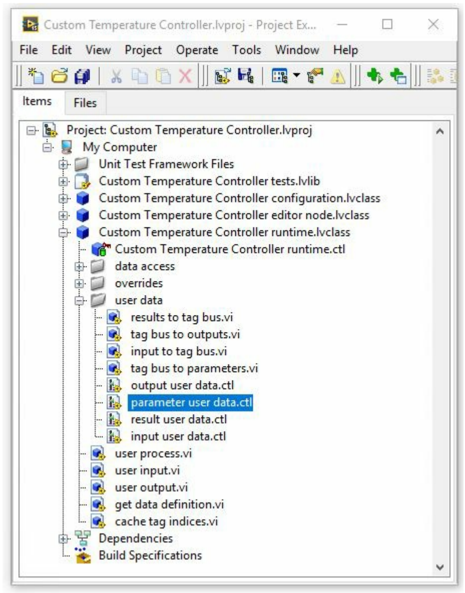
Part B: Update the Static PID Module with scripting

As w ith any project requirment changed and w e need to add channels for the P,I,D values. In this section w e are going to update our module channel list matches this table:

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

Update the clusters

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



Custom Temperature Controller.lvproj - Project Ex...

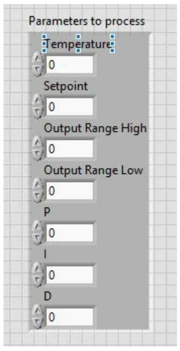
File Edit View Project Operate Tools Window Help

Items Files

```
Project: Custom Temperature Controller.lvproj
  My Computer
    Unit Test Framework Files
    Custom Temperature Controller tests.lvlib
    Custom Temperature Controller configuration.lvclass
    Custom Temperature Controller editor node.lvclass
    Custom Temperature Controller runtime.lvclass
      Custom Temperature Controller runtime.ctl
      data access
      overrides
      user data
        results to tag bus.vi
        tag bus to outputs.vi
        input to tag bus.vi
        tag bus to parameters.vi
        output user data.ctl
        parameter user data.ctl
        result user data.ctl
        input user data.ctl
      user process.vi
      user input.vi
      user output.vi
      get data definition.vi
      cache tag indices.vi
    Dependencies
    Build Specifications
```

Figure 1.6

11. Add the new channels for the cluster to look like in figure 1.7

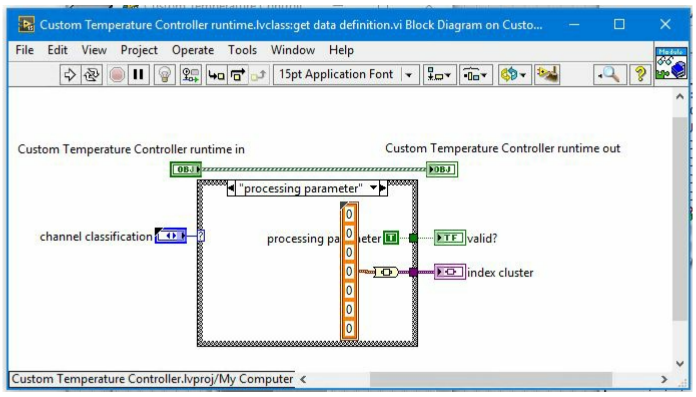


Parameters to process
Temperature 0
Setpoint 0
Output Range High 0
Output Range Low 0
P 0
I 0
D 0

Figure 1.7

12. 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 w hether or not the data interface is valid and if that method should be run. Setting a given Boolean to true tells the framew ork that you'd like to configure and run that method. If you follow ed the steps above, the Boolean should be true for processing parameters and processing results. The script w ill use these Booleans to determine if a given cluster is valid (the "placeholder" fields are just there to ensure the clusters are not broken).



Custom Temperature Controller runtime.lvclass:get data definition.vi Block Diagram on Custo...

File Edit View Project Operate Tools Window Help

15pt Application Font

Custom Temperature Controller.lvproj/My Computer

Figure 1.8

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 w hich currently have the class loaded. If you see a "lock" icon over the class, the script w ill not w ork.

13. Open **Tools >> DCAF >> Launch Control Module Scripting Utility...**
14. Drag the runtime class from the project over the runtime class path control or brow se for it manually, then repeat for the configuration class (YourModuleName Configuration.lvclass).
    Because you used the script, you can leave most options as the default and press Run. and it should look like Figure 1.9

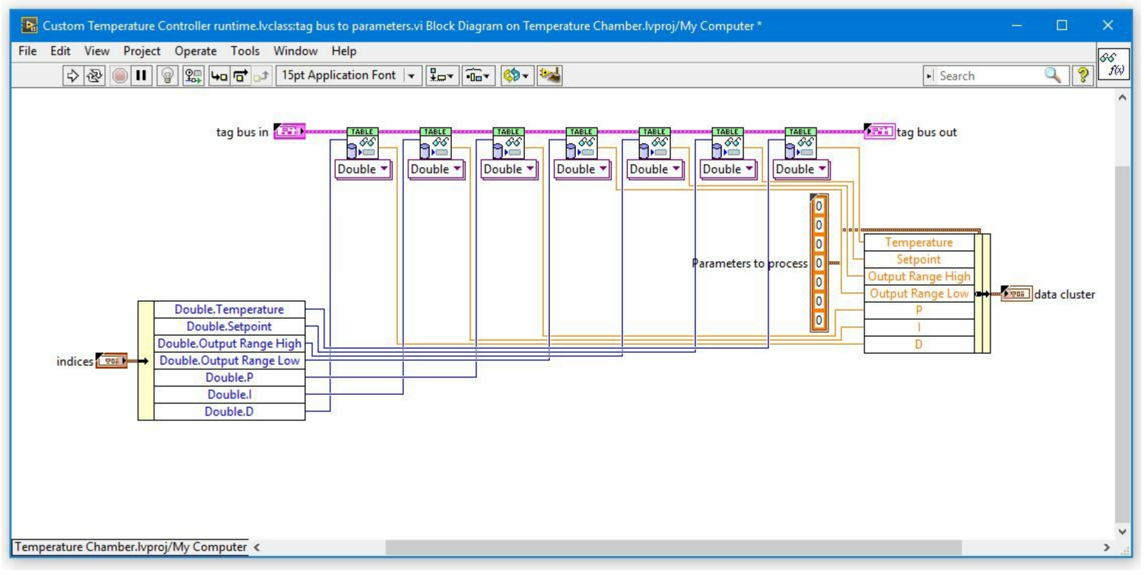


Figure 1.9

15. Save all the VIs

Part C: Implement the user process

16. Navigate to **Custom Controller Module runtime.lvclass**, open **user process.vi** and implement the same PID control as shown in the Figure 1.9

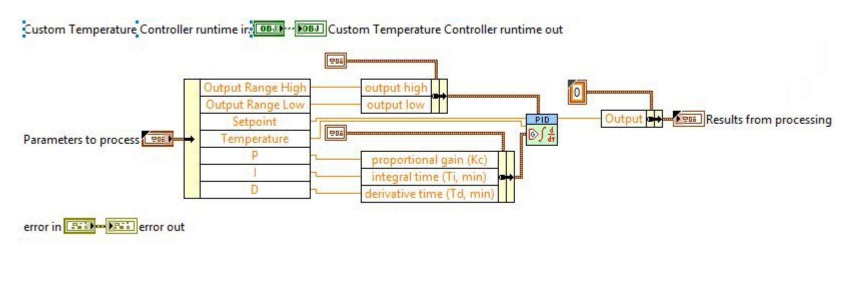


Figure 1.10

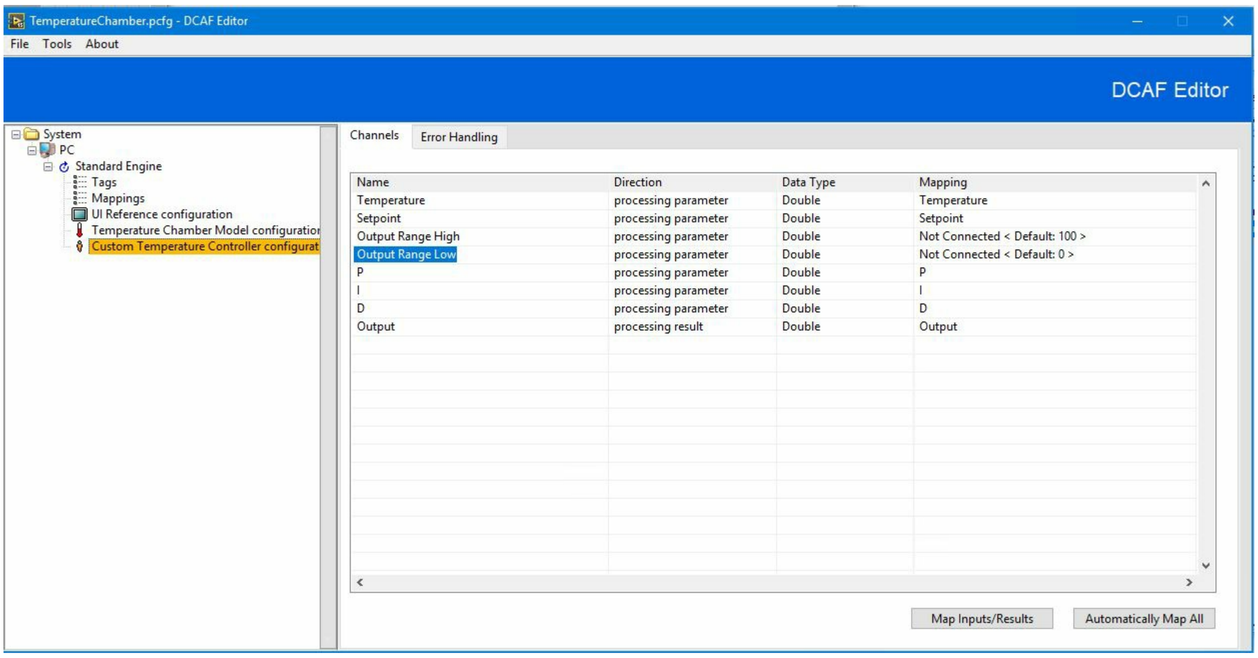
a. Drop down an instance of **PID.vi** from **Control and Simulation >> PID >> PID.vi.**
b. Drop down the following functions: 1 Unbundle by Name, 2 Bundle by Name function.
c. Wire the Unbundle by Name function to the Parameters to process cluster. Expand all the terminals.
d. Wire **Parameters to process.Temperature** to **PID.vi process variable input**.
e. Wire Parameters to **process.setpoint** to **PID.vi setpoint input**.
li> Create a constant from **Results from PID.vi output range input**. and wire it to **Bundle by Name**.
f. Bundle Parameters to **process.output range high** and **Parameters to process.output range low ** and wire the cluster to **PID.vi output range input.**
li> Create a constant from **PID.vi PID gains input**. and wire it to **Bundle by Name**.
g. Bundle **Parameters to process.P**, **Parameters to process.I**, and P**arameters to process.D** and wire the cluster to **PID.vi PID gains input**.
h. Create a constant from Results from processing indicator and connect it to the Bundle by Name input cluster terminal. Expand the Unbundle by Name Function to have 2 terminals.
i. Wire PID.vi output to **Output** from **processing.Output**.
j. The result should look something like Figure 1.9.

17. Save the new project and close it.

Part C: Running and Configuring the Module

Configuration Editor

18. Open Standard Configuration Editor (**Open Tools >> DCAF >> Launch Standard Configuration Editor…**).
19. Navigate to **Tools >> Edit Plugin Search Paths**.
20. Press Add and navigate to the location of your new control module (**Temperature Controller\Modules**).
Note: Add the whole modules directory as the simulation module is in that folder.
21. Open Static **Temperature Controller.pfcg** in the exercise folder.
22. Now, right click on Standard Engine and select **Add >> Other>>Custom Controller Module**. Then, select this new module from the tree.
23. 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 Figure 1.11:



TemperatureChamber.pcfg - DCAF Editor

File Tools About

DCAF Editor

- System
  - PC
    - Standard Engine
      - Tags
      - Mappings
      - UI Reference configuration
      - Temperature Chamber Model configuration
      - Custom Temperature Controller configurat

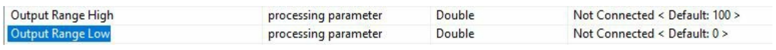
Channels | Error Handling

Name	Direction	Data Type	Mapping
Temperature	processing parameter	Double	Temperature
Setpoint	processing parameter	Double	Setpoint
Output Range High	processing parameter	Double	Not Connected < Default: 100 >
Output Range Low	processing parameter	Double	Not Connected < Default: 0 >
P	processing parameter	Double	P
I	processing parameter	Double	I
D	processing parameter	Double	D
Output	processing result	Double	Output

Map Inputs/Results | Automatically Map All

Figure 1.11

18. Now, 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.



Output Range High	processing parameter	Double	Not Connected < Default: 100 >
Output Range Low	processing parameter	Double	Not Connected < Default: 0 >

Figure 1.12

19. Finally, we need to map our results to system tags.
    Note: If the tag we need already has a writer you have to disconnect it.
20. Select to File >> Save.
21. From the Temperature Controller Project Window, open Host Module Includes.vi. 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.

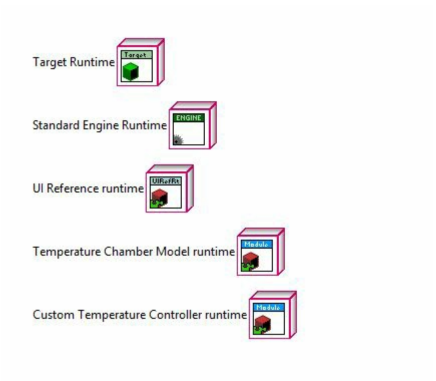
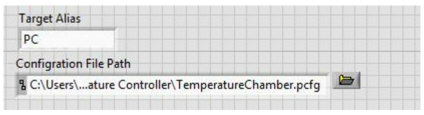


Figure 1.13

Running the Code

24. Open the **Temperature Chamber** Project located in **\Temperature Controller**
25. Verify the File path is the same as the one you use to store the configuration.



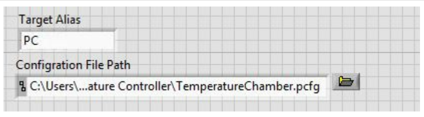
Target Alias

PC

Configration File Path

C:\Users\...ature Controller\TemperatureChamber.pcfg

Figure 1.13



Target Alias

PC

Configration File Path

C:\Users\...ature Controller\TemperatureChamber.pcfg

Figure 1.13

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

26. Run the **Main.VI**

Exercise 2:

In this exercise we will create a DCAF Static module that contains a state machine. This is a optional and advance 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 w ill be a simple state machine that contains 3 states. This states w ill transition from one state to the next only if just the correct Boolean is selected.

The state machine w ill have simple transitions.

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

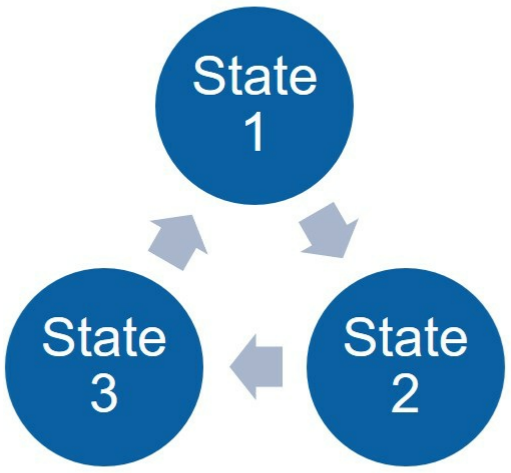


Figure 2.1 State Machine Diagram.

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 w ill 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 the state to the runtime
3. Initialize the state
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 w ith 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.

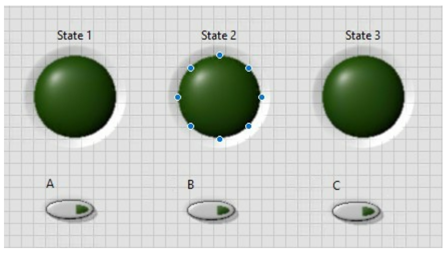


Figure 2.2 State Machine UI.