Hands-On 1: Creating a DCAF Application

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 ni.com/dcaf

Setup

To install DCAF in LabVIEW open VI Package Manager (VIPM), search for DCAF, and install in the corresponding LabVIEW version. This package can be installed to any LabVIEW version from 2015 to present. A CompactRIO controller is not required.

Introduction

Most control applications have similar challenges and needs. By working with different large control applications, we managed to identify most of this common challenges and needs and created DCAF to provide a standard framework to develop control applications.

Most of these challenges are related to having different processes running in parallel that need to share data without falling into race conditions. DCAF provides the capability of creating synchronized engines to run standard and custom modules and defining the mapping of data between them through a simple interface known as the Configuration Editor.

Before we start with the Exercises we need to understand some basic terminology of how a DCAF system is structured.

System: Your System will consist of one or more targets containing the one or more **DCAF Engines**.

Target: A **Target** will represent the physical device that will run one or more **Engines**. A **Target** could be a PC or a CRIO.

Engine: The Engine will be in charge of executing **Modules** in a synchronous way and transfer data between them through the **Tag Bus**.

Module: Piece of code with a specific functionality that will be executed within an Engine. Some standard Modules are installed with DCAF, but you can create your own modules.

Once we have defined the terminology to understand the hierarchy of a DCAF system, we need to understand how the data flows through a DCAF system. Data can be passed between **Modules, Engines** and even **Targets**. Here is some more terminology related to dataflow in DCAF.

Channels: Parameters that allow access to and from a **Module**. Channels can be Inputs, Outputs, Processing Parameters and Processing Results – note that the direction is taken to be from the engine's point of view.

Tags: Scalar variables saved in a single repository (**Tag Bus**) that can be accessed by any **Module** within an **Engine**. A **Tag** can be defined as a connecting point between **Channels** from different **Modules**.

Mappings: Mappings are the connections between **Tags** and **Channels**. If you want a specific Channel to write or read a value

on a specific **Tag** you will have to map them.

Take the following example to clarify the previous terminology. Let's say a Module called **Temperature Chamber Model** has an Input Channel called T**hermocouple Reading** – the module implements reading from a thermocouple and puts the value into the **Thermocouple Reading** channel. This **Thermocouple Reading Channel** is mapped to a Tag called **Temperature** – the engine will then take the value that the module places onto the channel and put it on the tag. Then the **Temperature** Tag's value is passed by the engine to Temperature, an Output Channel that belongs to a module called **Temperature Controller Logic**.
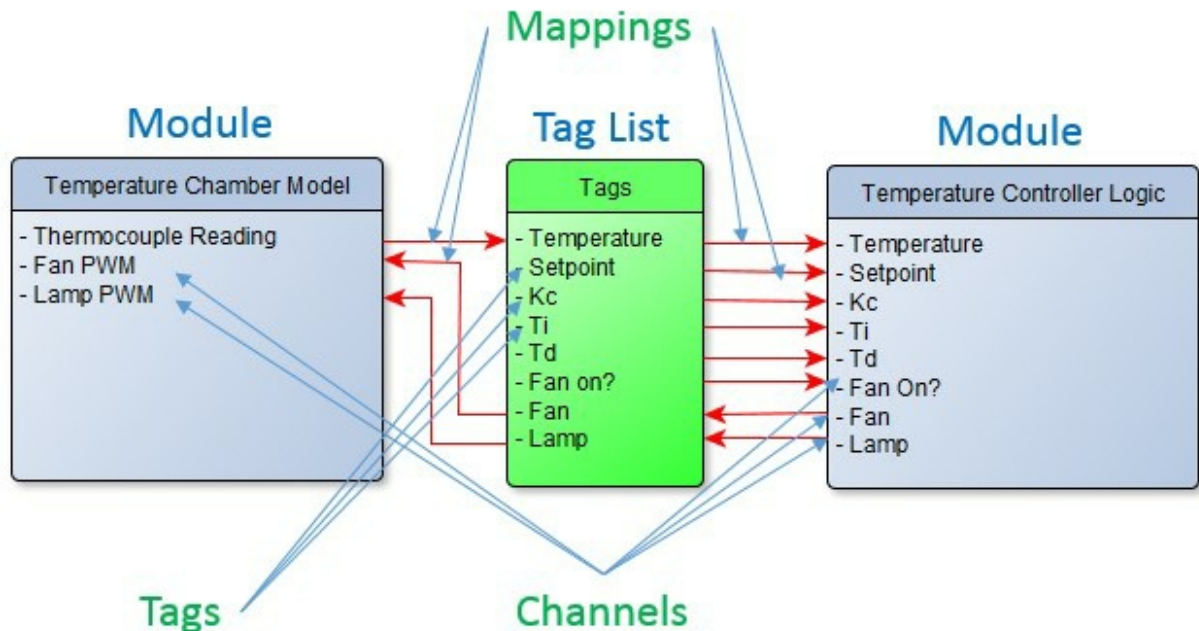
Exercise 1:

This exercise demonstrates the implementation of a simple temperature chamber controller application. It 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 within DCAF to provide communication between modules. You will also learn how to create a UI and map it to data within the framework.

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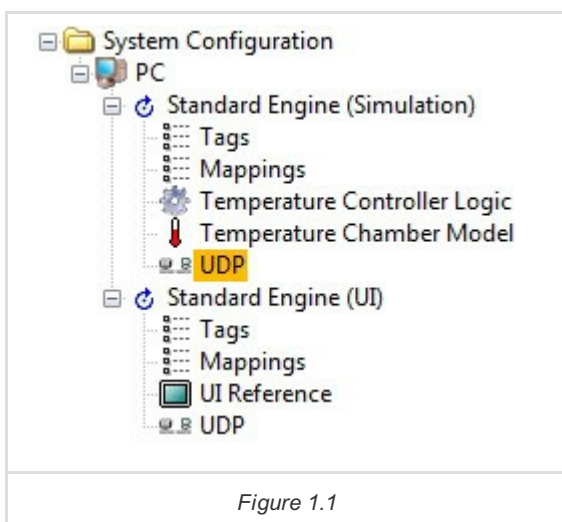

Figure 1.1

In the hierarchy shown above you can find some of the components defined in the previous section. In each of these **Engines** you will find **Modules**. Some of these **Modules** are standard and some of them were created specifically for this Hands On.

Standard Modules

UDP: This module exists in both components. It is designed to share tags between Engines by mapping each tag as an Engine Input or Output. All 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 takes a pre-existing front panel and maps its controls and indicators to DCAF tags to permit direct user interaction with the framework.

Custom Modules

Temperature Controller Logic: This is a custom DCAF Module designed to provide the control logic for the temperature chamber. 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 A: Project Creation and UI

During this first part of the exercise you will be create a DCAF project from scratch using a template and learn how to add a User Interface to your DCAF project.

1. In LabVIEW go to **File >> Create Project..** and select DCAF. From the displayed list select **Basic Execution Template** and press the **Next** button.
2. Name the project **Temperature Controller** and select **\Desktop\Hands On\Hands-On_1\Exercises\Temperature Controller\Runtime\Project** as the Project Root.
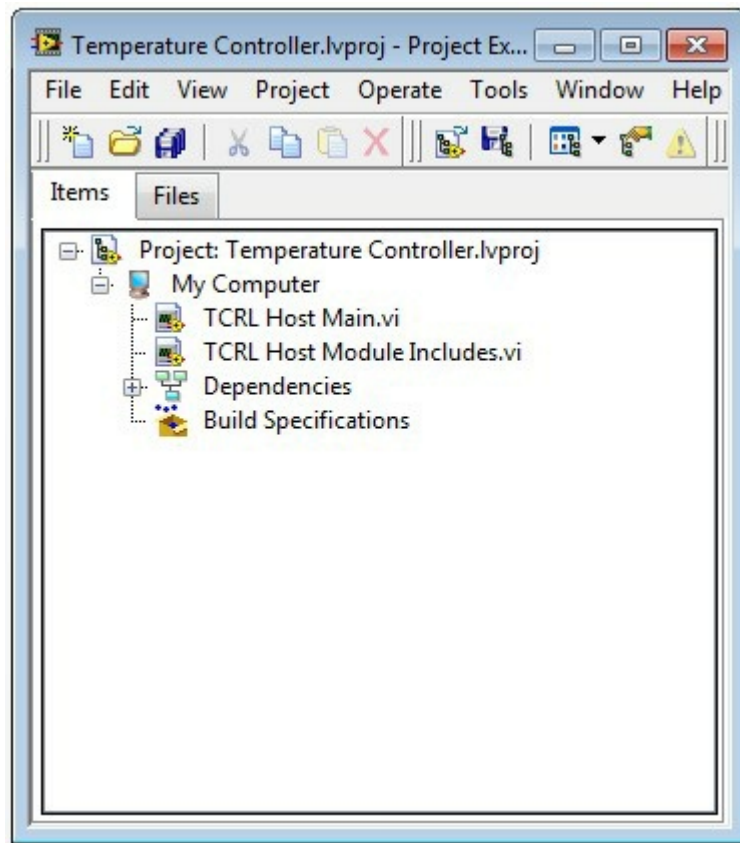3. Verify your project window matches Figure 1.2.
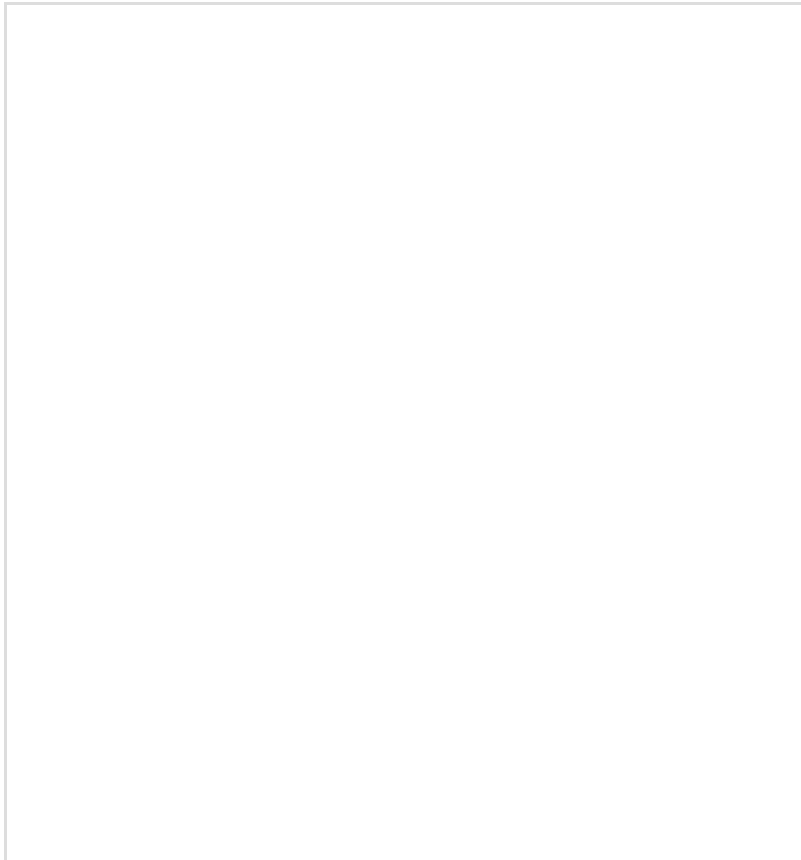
Figure 1.2

4. Add to the project a Configuration File with the Engines for the Temperature Controller partially configured and mapped (in future exercises you will make a Configuration File). Under **My Computer** add **SimulatedSystem.pcfg** located at **\Desktop\Hands On\Hands-On_1\Exercises\Temperature Controller**.

5. To speed up the exercise, a UI has already been created. Under **My Computer**, add **TCRL User Interface.vi** to the project located at **\Temperature Controller\Runtime.**

6. Look at 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. Save and Close **TCRL User Interface.vi**.
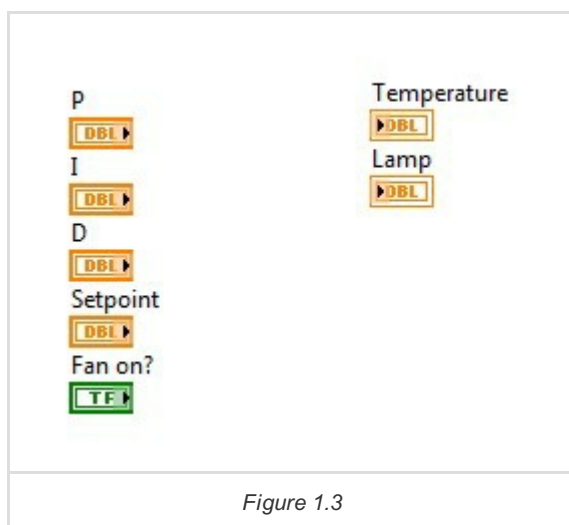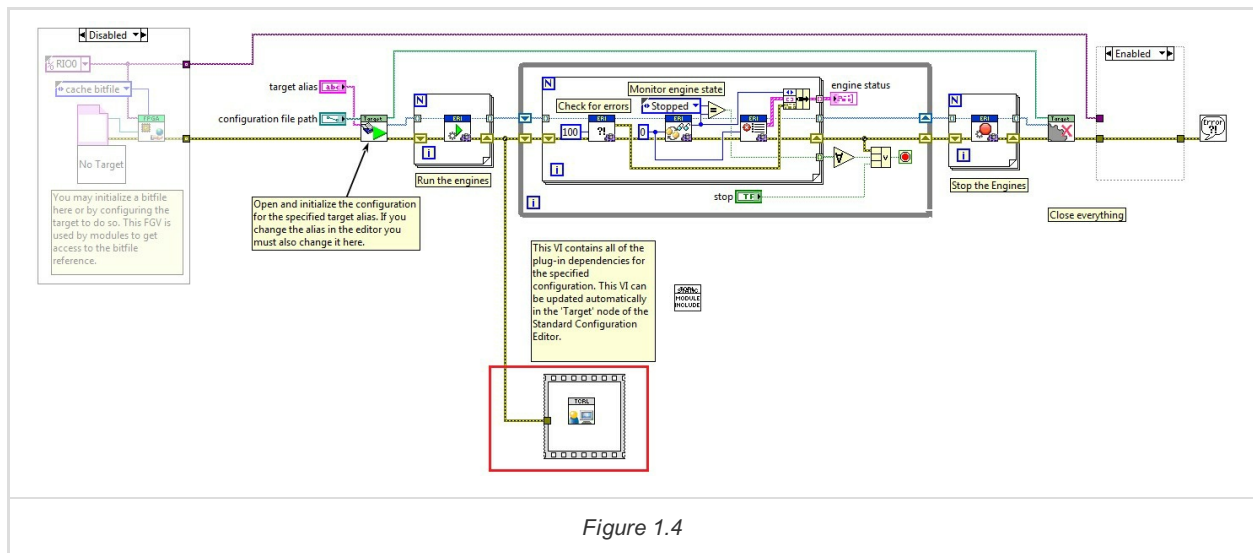

Figure 1.3

7. Open **TCRL Host Main.vi** Block Diagram. Delete the bottom While Loop, as we won't need it for this exercise. 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 shown in Figure 1.4.

Figure 1.4

8. Open **TCRL Host Main.vi** Front Panel. In the configuration file path control browse for **SymulatedSystem.pcfg** located at **\Temperature Controller**. Select this as default value for this control by going to **Edit >> Make Current Values Default**. Save and Close this VI.

Part B: Adding Required Classes

DCAF has been developed using LabVIEW Object Oriented Programming. Therefore, the code will only run if the classes used within a specific configuration are added to the project. DCAF provides a simple script that will help you with this every time you add or remove modules to a target in the Configuration Editor. This is not automatic, so you have to remember to run this scripting tool when you make these kinds of changes in the configuration.

1. Open the Standard Configuration Editor by navigating in LabVIEW to **Tools>>DCAF>>Launch Standard Configuration Editor…**
2. Navigate within the editor to **Tools>>Edit Plugin Search Paths**.
3. Add a search path to the plugins for this example located at **\Temperature Controller\Modules** 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 **\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\Hands On\Hands-On_1\Exercises\Temperature Controller\Runtime** if not already opened.
7. Open **TCRL Host Module Includes.vi** and verify the Block Diagram is empty. This VI will load the required classes when T**CRL Host Main.vi** executes. A scripting tool will add the corresponding classes to TCRL Host Module Includes.vi. In the System Configuration hierarchy in the DCAF Configuration Editor select PC. In the Includes file path box browse for **TCRL Host Module Includes.vi** located at **\Temperature Controller\Runtime\Project**. Press the Generate button. Repeat this step each time you add or remove any Module from the Hierarchy Tree.

Figure 1.5

8. Verify that the corresponding classes have been added to **TCRL Host Module Includes.vi** and compare them to Figure 1.6:

Standard Engine Runtime

Temperature Controller Logic runtime

Temperature Chamber Model runtime

UDP Broadcaster Runtime

UI Reference runtime

Figure 1.6
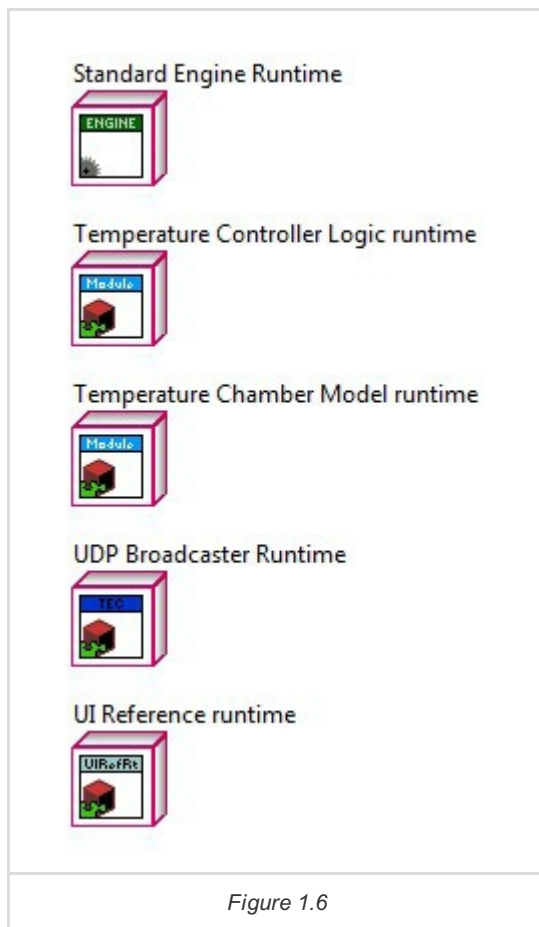
9. Save and close **TCRL Host Module Includes.vi**.

Part C: Mapping Tags in the Configuration Editor

This DCAF project has 2 engines: **The Simulation Engine** and the **UI Engine.** Both engines have listed Tags, Mappings, and UDP items. The rest are specific modules for each engine.

- The **Tags** item refers to the list of tags in the Tag Bus for each engine.
- The **Mappings** item allows to configure and visualize the connections between each **Module** parameter (Input/Output) and the Tag Bus.
- The **UDP** item publishes tags that can be shared with another engine that might be in the same target or in a different one. During this last part of the exercise you will learn how to map **Tags** between Modules through the Tag Bus in each Engine and share Tags between Engines through UDP.

Before we start the implementation, take a look to the following diagram to understand how data flows through Modules and Engines of our Simulated Temperature Controller.
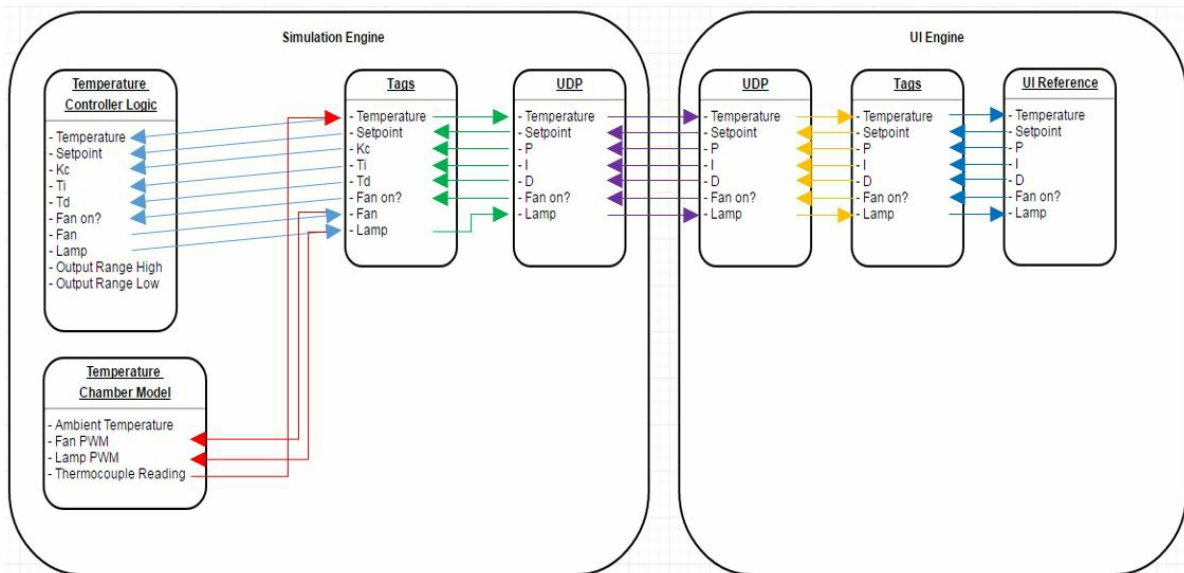
Figure 1.7

1. Open the Standard Configuration Editor by navigating in LabVIEW to **Tools>>DCAF>>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. Browse for **TCRL User Interface.vi** located at **\Temperature Controller\Runtime**.
3. Press the **Configure from UI** button. When the pop up asking to Automatically map tags to channels appears select Yes. Verify your mappings comparing them with Figure 1.8.
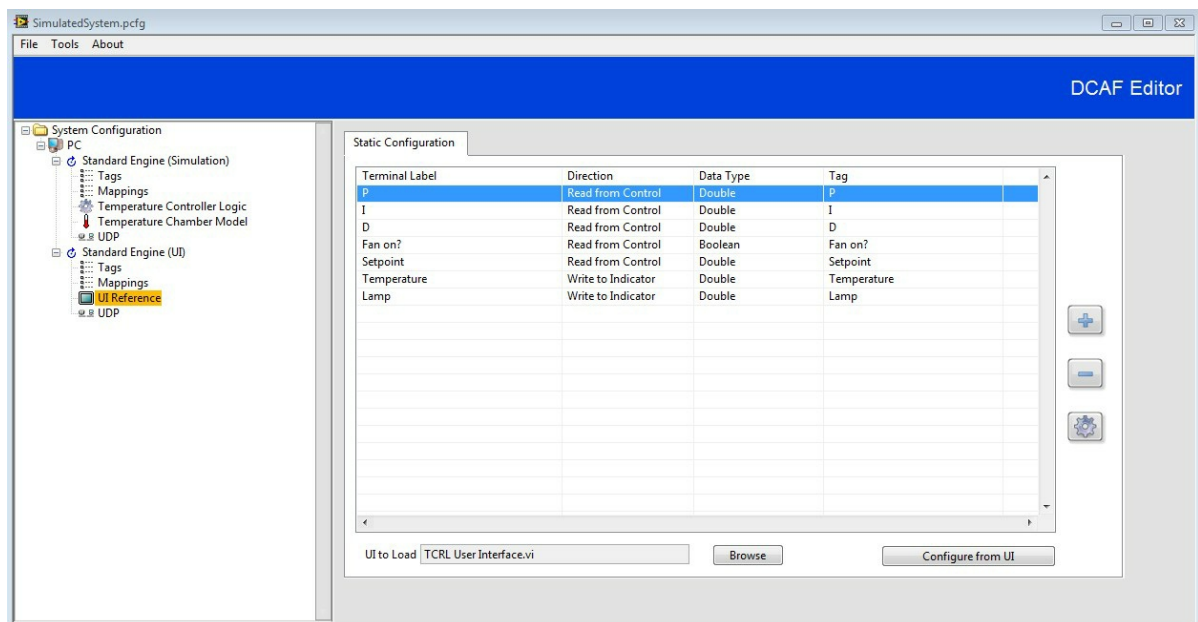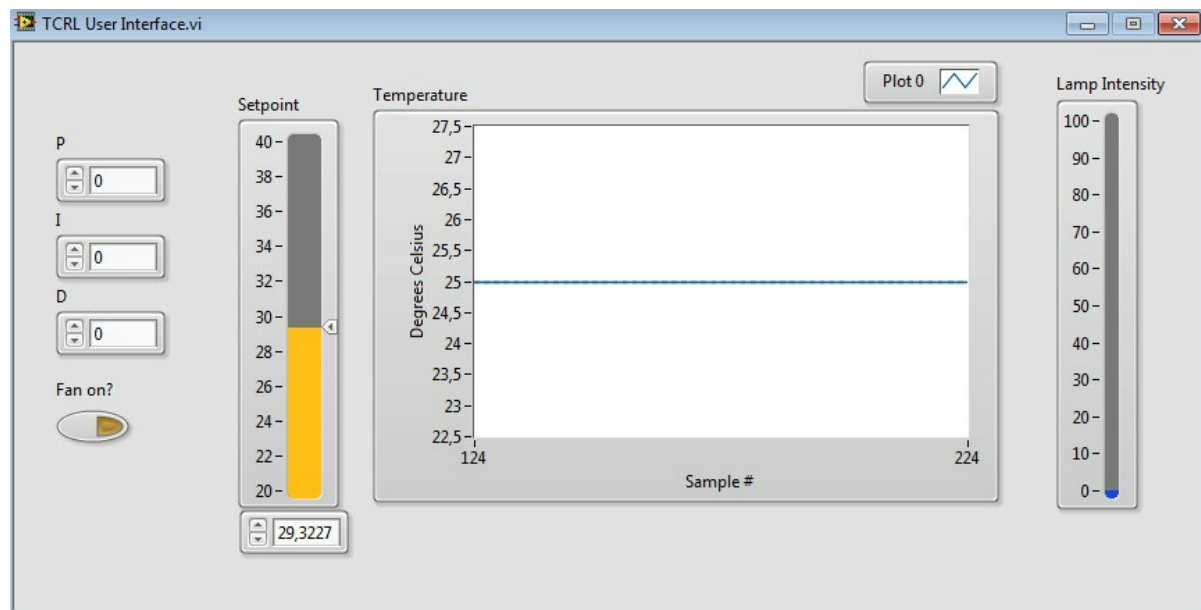

Figure 1.8

4. Save the changes in the Configuration Editor.
5. Open and run **TCRL 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

Note: 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.

6. Stop the **TCRL Host Main.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.

7. We will start with the **Simulation Engine**. First select the Tags node and take a look at the tags.

Figure 1.10
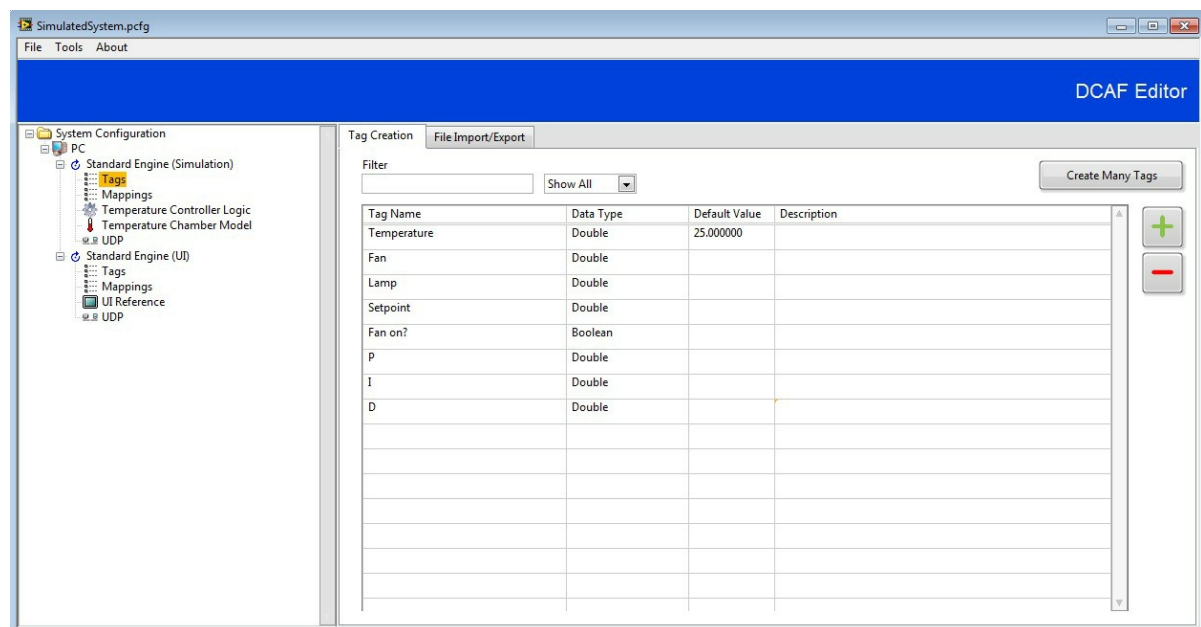
8. These tags are used for connections in the rest of Simulation Engine modules: Temperature Controller Logic, Temperature Chamber Model and UDP. Notice all of them are Doubles except for **Fan on?**.

9. Go to **Mappings** under the Simulation Standard Engine 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 look, don't make changes.
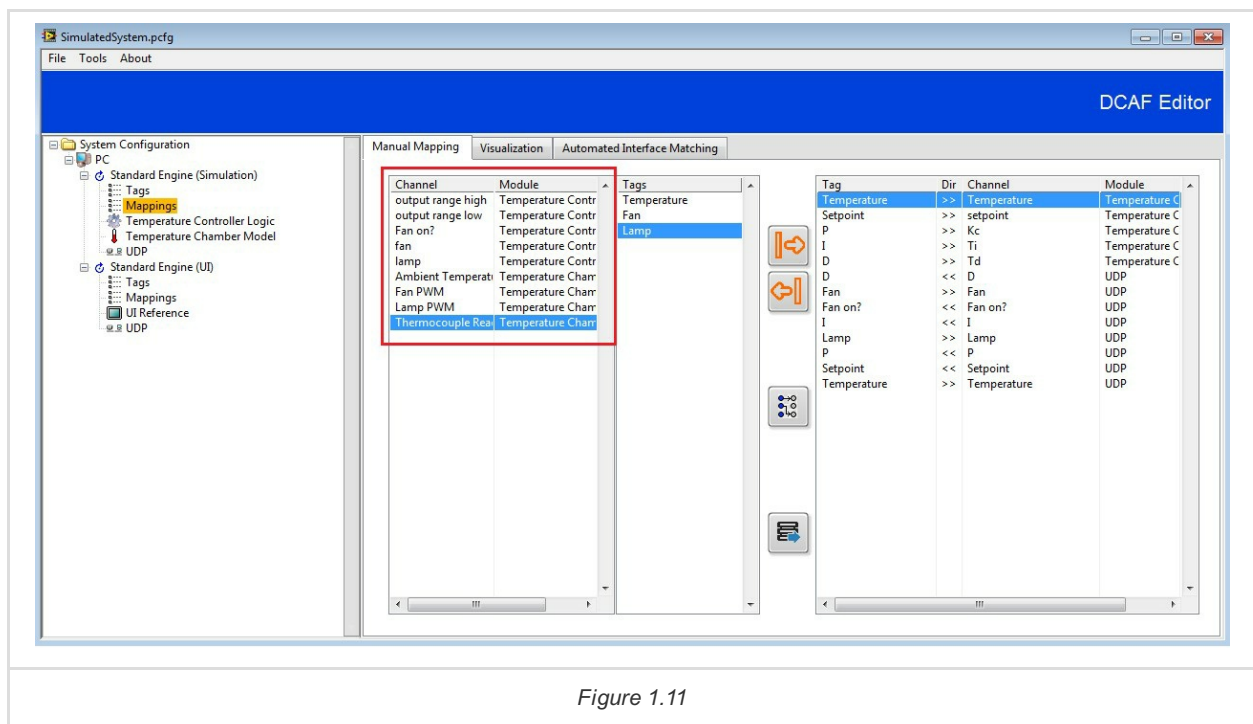
Figure 1.11

10. 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**. These are internal variables with constant values defined statically. The rest should be mapped to a tag.

11. 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, left click, and select the corresponding tag from the **Available Tags** list.
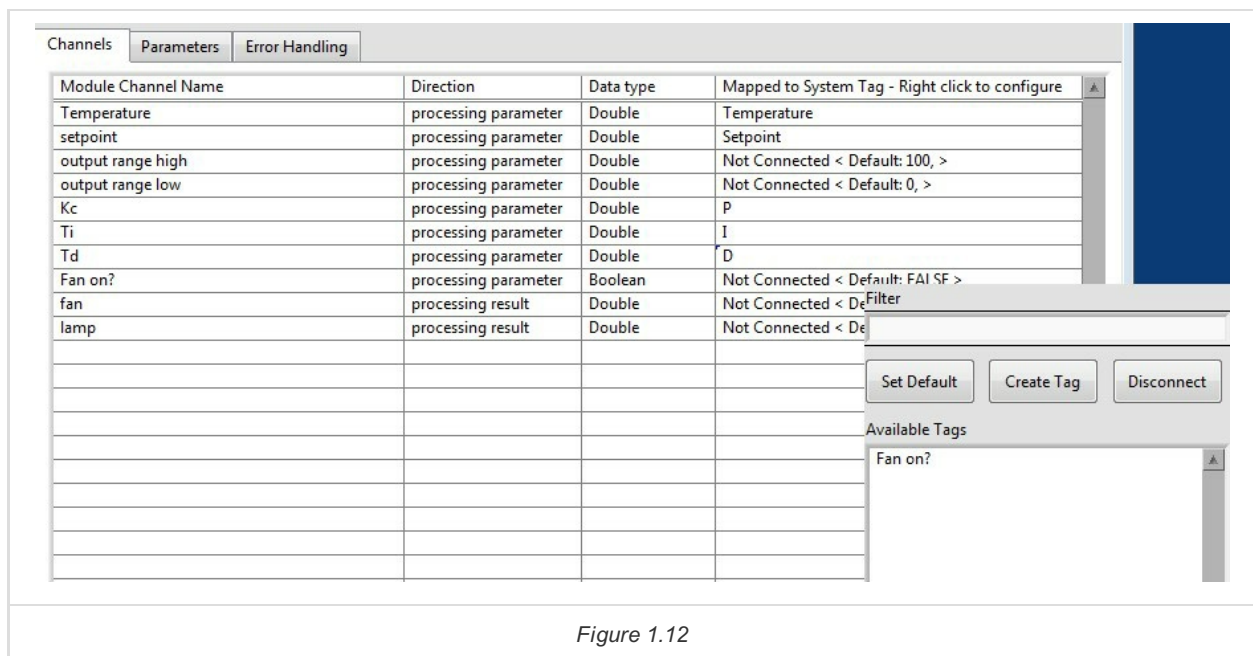

Figure 1.12

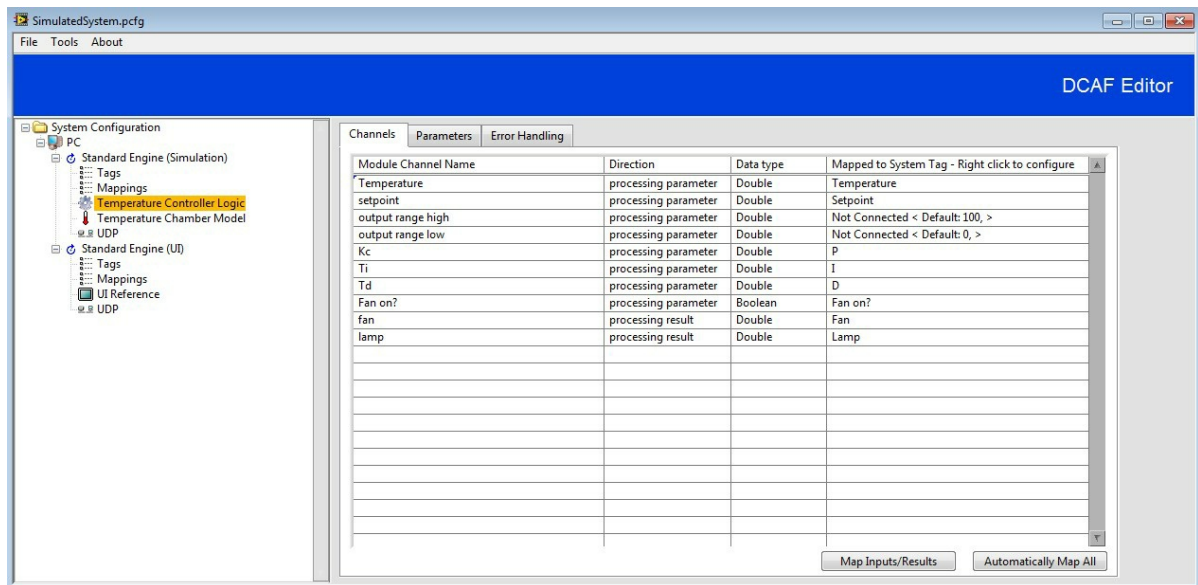12. Verify your table looks like the following image:

Figure 1.13

13. 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.

14. Go to the **Temperature Chamber Model** module. Notice all the channels are disconnected from any tag. The only disconnected channel should be **Ambient Temperature**. Create the following connections. **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 11, map **Fan PWM**, **Lamp PWM**, and T**hermocouple Reading** channels to **Fan**, **Lamp**, and **Thermocouple** tags. Verify your table looks like the following image:
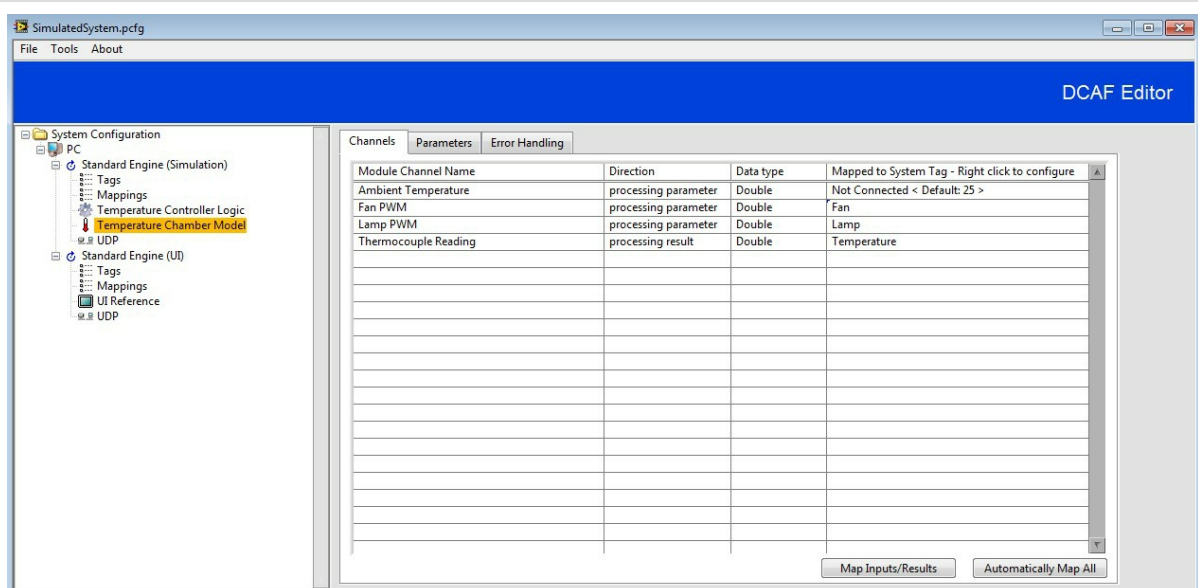

Figure 1.14

15. 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

16. 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
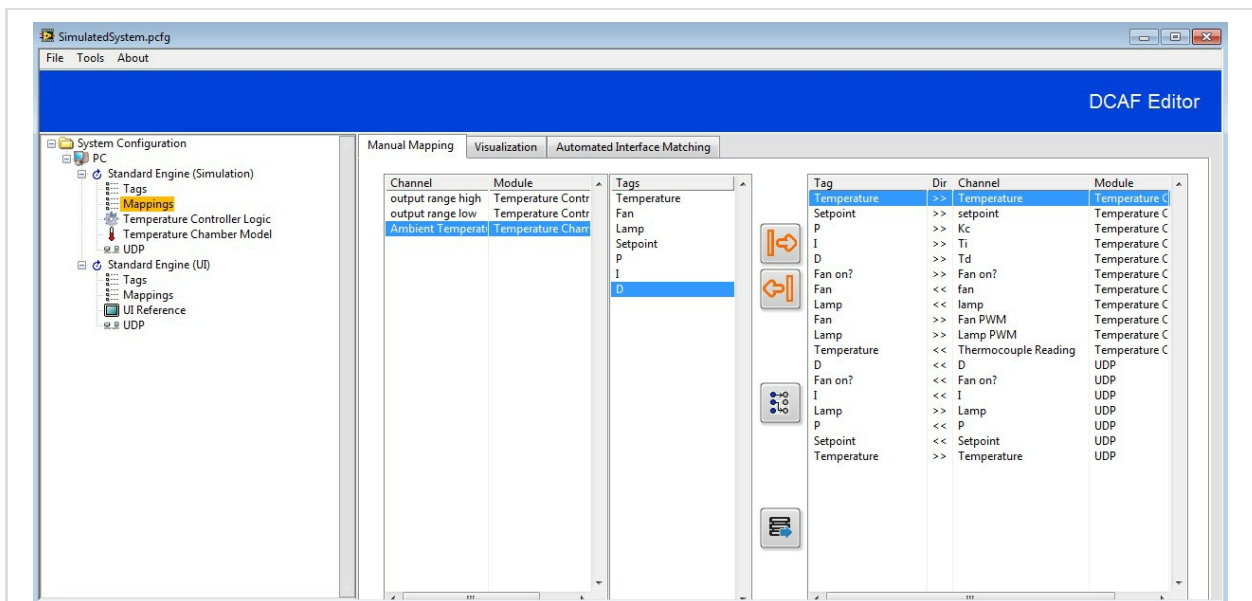
17. Now that you have seen how the **Simulation Engine** works, step into the different components of the UI Engine to understand how it interacts with the **Simulation Engine**. Notice that the Inputs for the **UI Engine UDP Module** are the Outputs for the **Simulation Engine UDP Module** and vice versa.
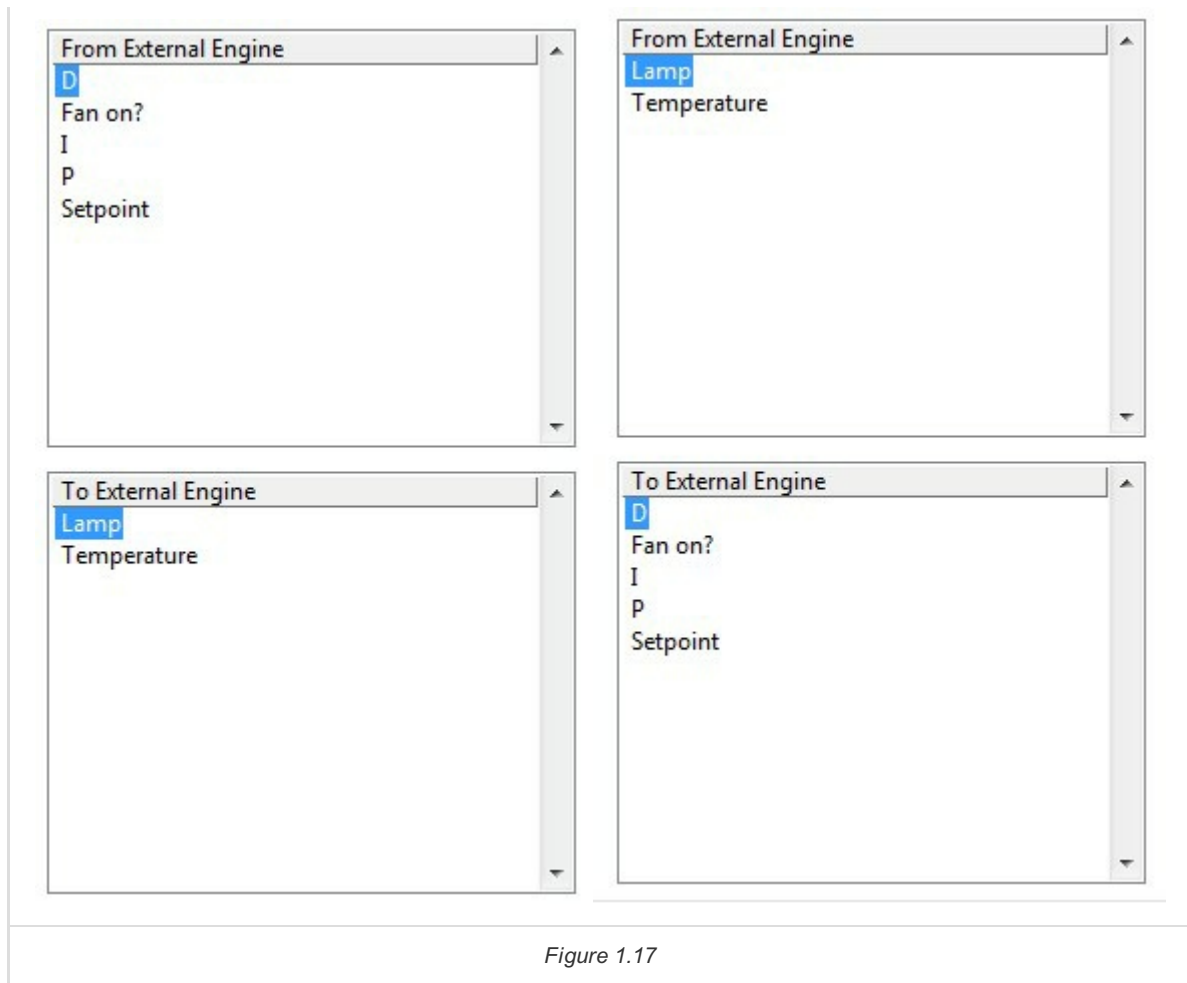
Figure 1.17

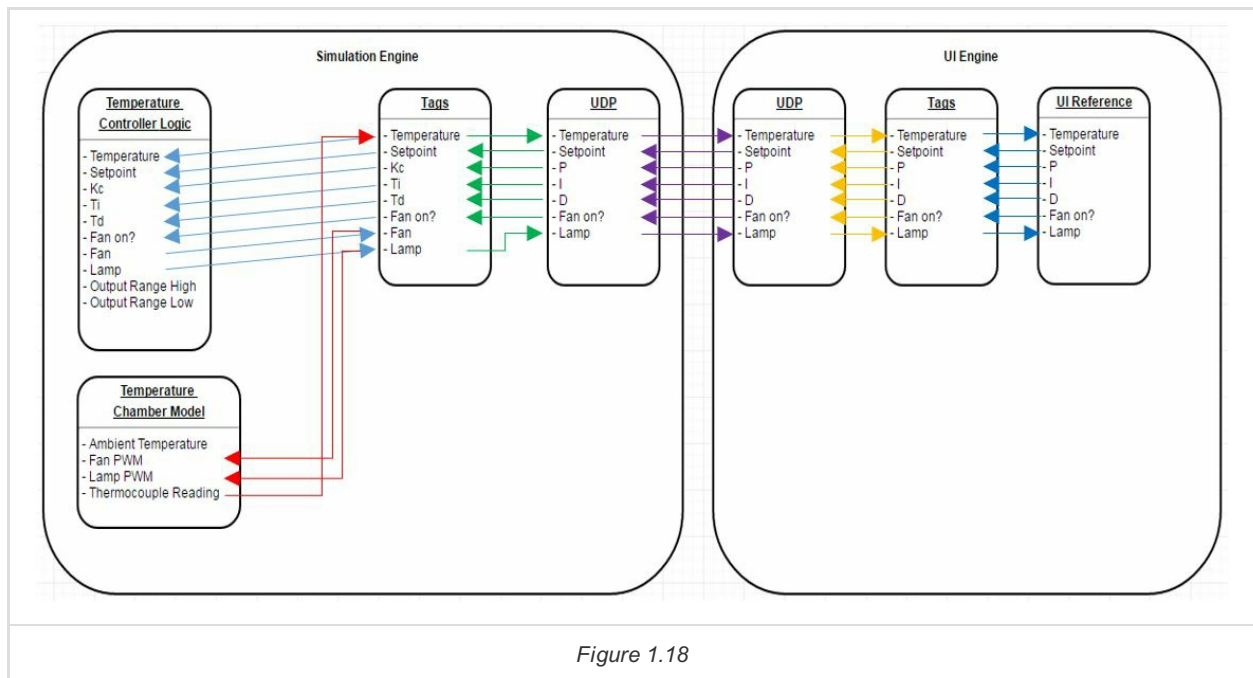18. Take a look again to the dataflow diagram to review the mapping you just did.

Figure 1.18

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

Figure 1.19
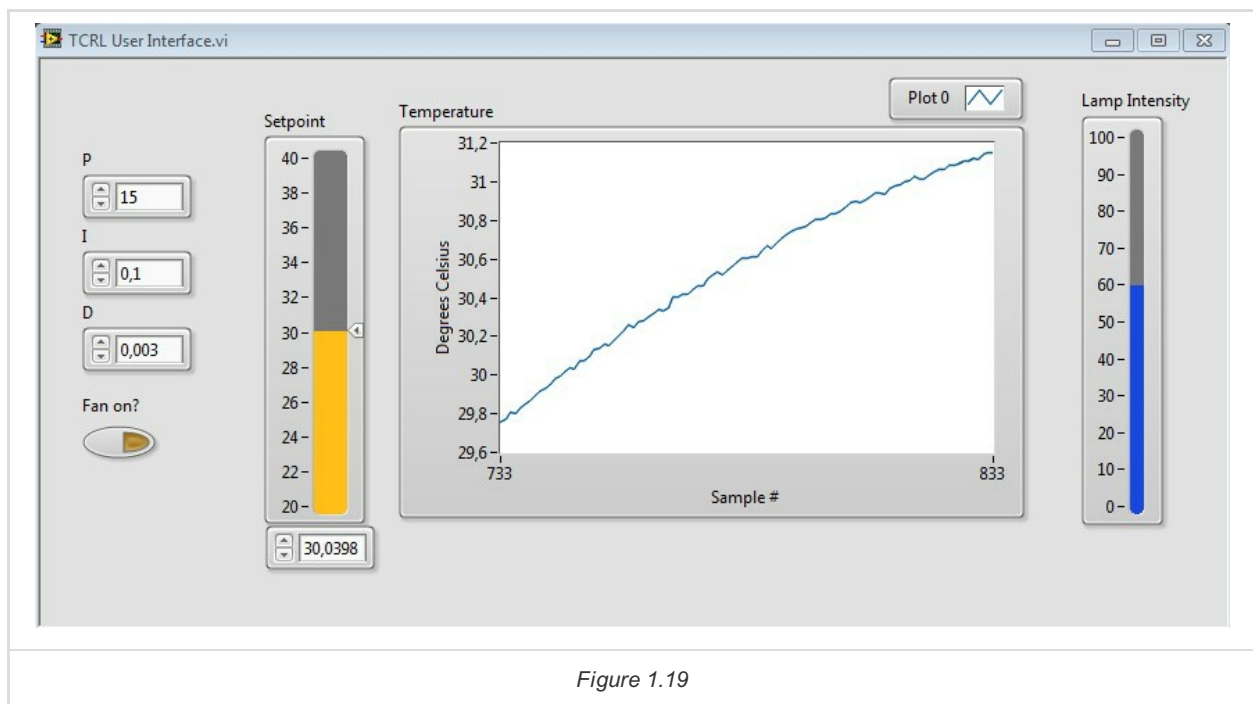
Exercise 2: Adding Standard Modules to the Temperature Control Application (TDMS & CVT)

In Exercise 1 you developed a Simple Temperature Control Application using DCAF. Now, we will add standard features such as TDMS and CVT to learn how to add standard DCAF modules to your application.

Part A: Add TDMS

Adding TDMS is a specific module that might become really handy in a DCAF application. This part of the exercise will guide you through the process of adding TDMS logging to your DCAF application.

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>>Utilities>>TDMS datalogger** as shown in Figure 2.1.
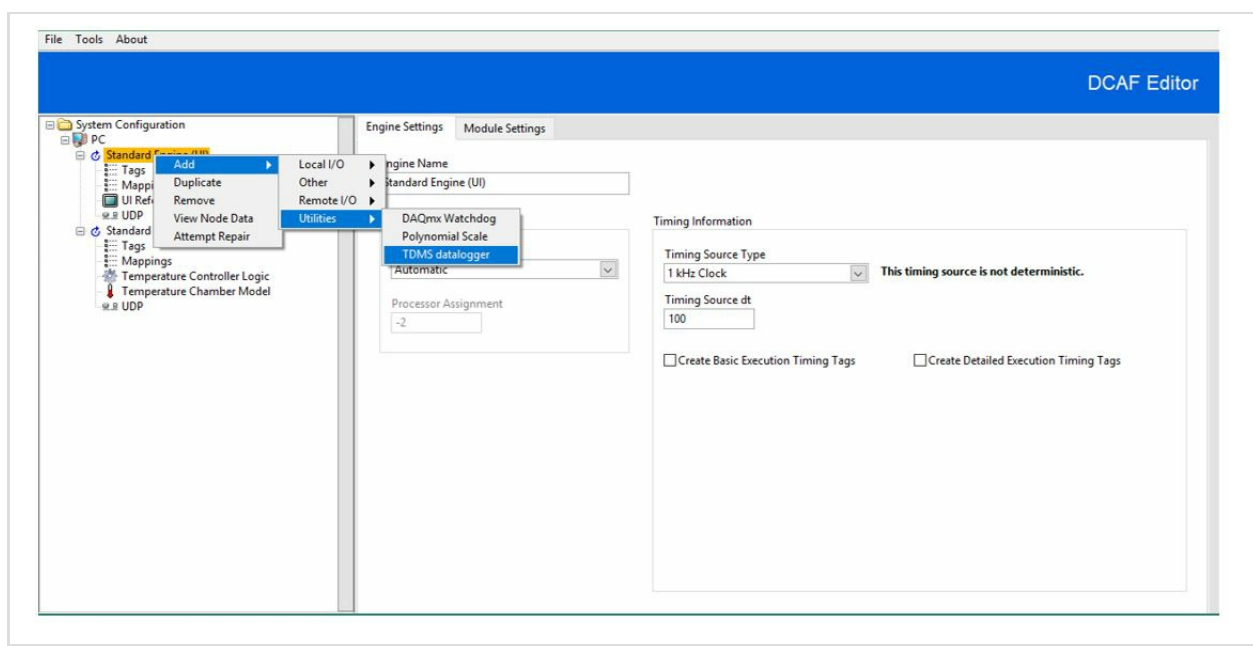
Figure 2.1

4. Select the **TDMS datalogger** item you just created. In the Static Configuration tab move Temperature, Setpoint, P, I, and D to the **Configured to Log** box.
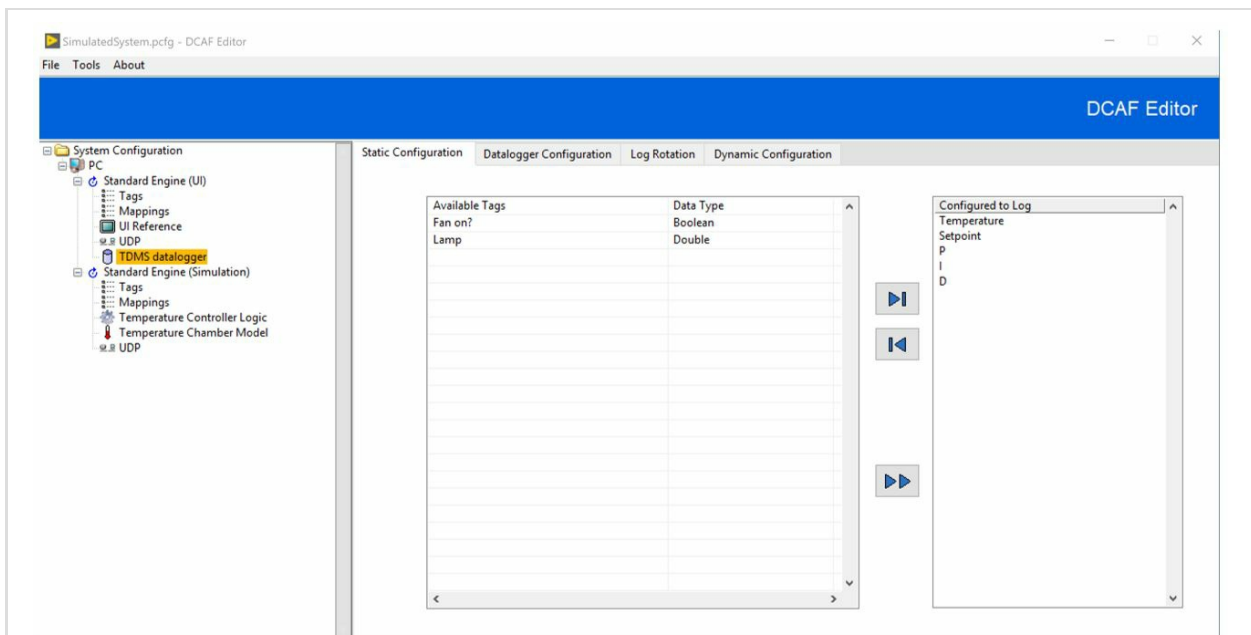

Figure 2.2

5. Go to the **Datalogger Configuration** tab. Press the brow se button to select a File path. Brow se to **\Temperature Controller\Runtime** and create a folder named **Data**. Type Temperature Measurements as the File name.
6. Go to the Log Rotation Tab. In here w e can configure a maximum size for each file before a new one is created. Press the brow se button to select a Historical Directory. Brow se to **\Temperature Controller\Runtime** and create a folder named **Historical**. In this folder our files w ill be saved once they get to the maximum size w e configured.
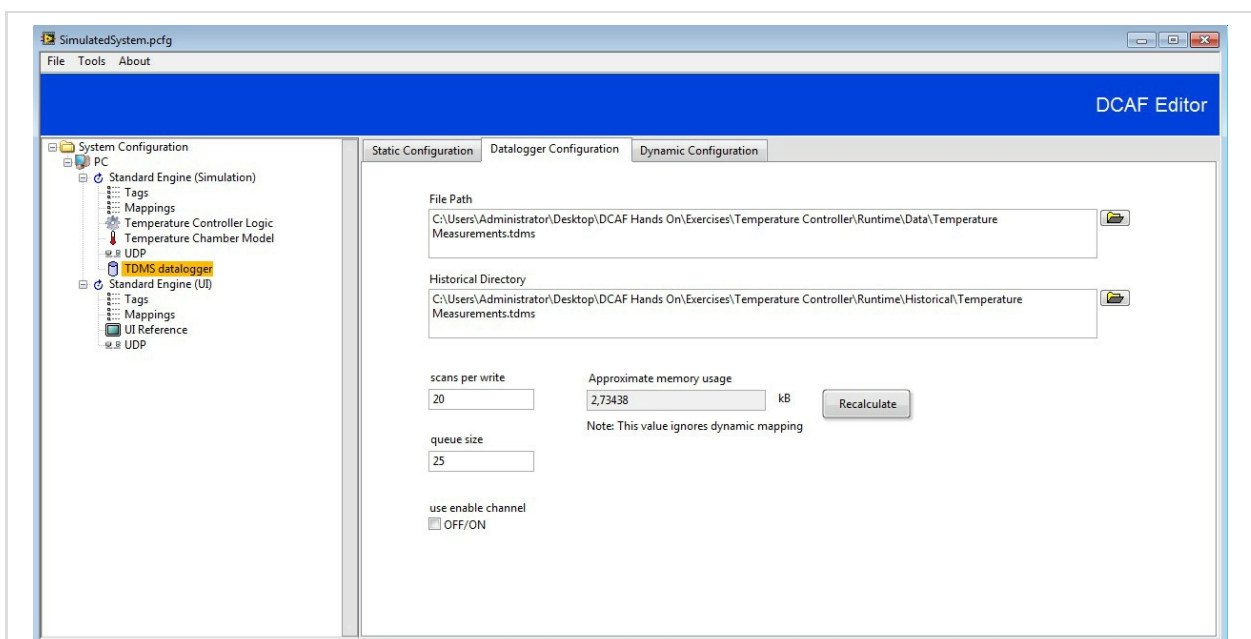

Figure 2.3

7. Since w e 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**. The option to re-generate the Includes VI is in the PC Configuration. Verify the class has been successfully added to **TCRL Host Module Includes.vi**.
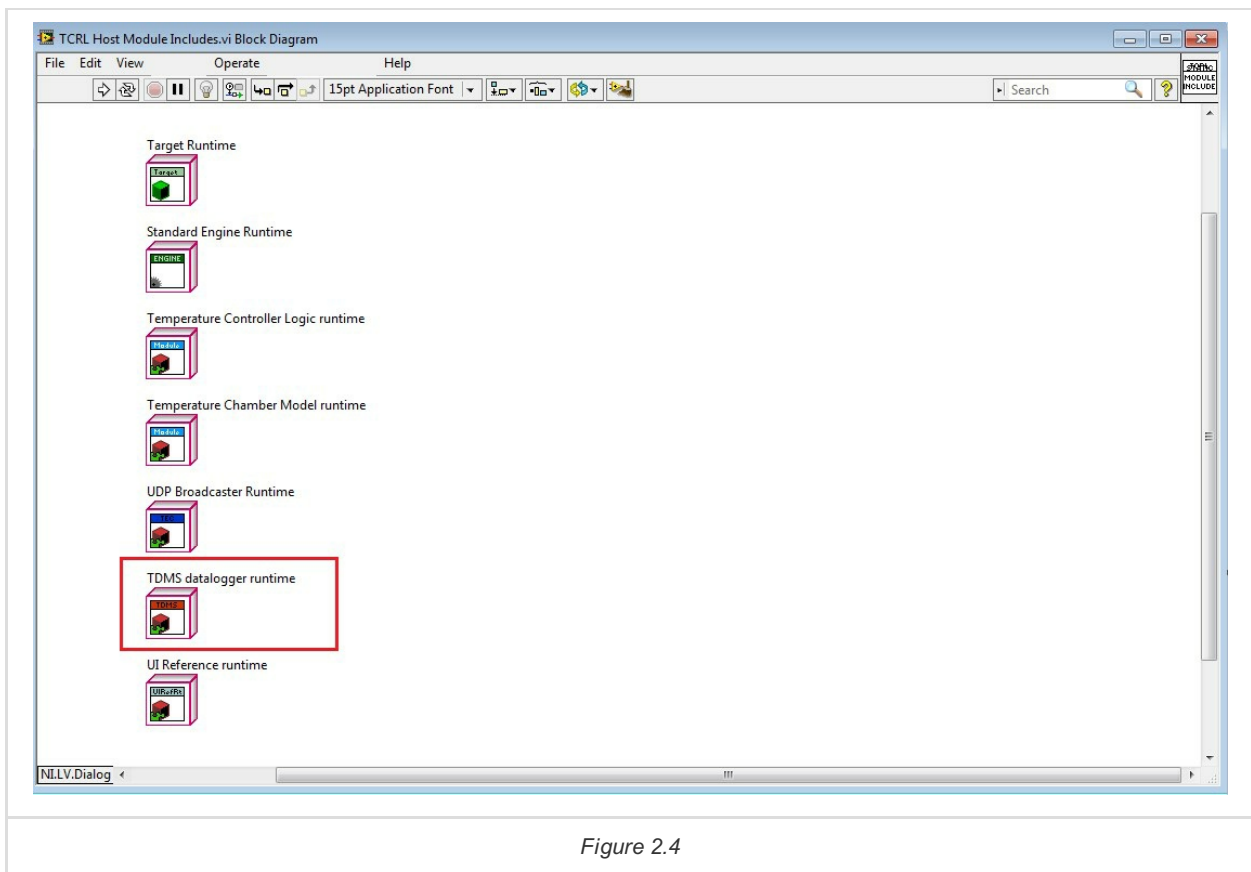

Figure 2.4

8. **Run TCRL Host Main.vi**. Do some changes to the setpoint and verify it still w orking and stop the VI.
9. Go to the Historical folder you created located at \Temperature Controller\Runtime\Historical and open the TDMS file just created. Verify the tags you added in the TDMS datalogger modules appear in the file and generated data.

   Note: if the file is not in **Historical** folder, the configuration migth be w rong or the file is open and it is still on the **Data** folder.

|	A	B	C	D	E	F
1	Temperature	Setpoint	P	I	D	Sample Time
2	25	0	0	0	0	09/05/2016 07:38:10,982 p.m.
3	25	25	15	0	0	09/05/2016 07:38:11,082 p.m.
4	25,01166054	25	15	0,1	0,003	09/05/2016 07:38:11,182 p.m.
5	24,99352431	25	15	0,1	0,003	09/05/2016 07:38:11,282 p.m.
6	25,01347714	25	15	0,1	0,003	09/05/2016 07:38:11,382 p.m.
7	24,99465394	25	15	0,1	0,003	09/05/2016 07:38:11,482 p.m.
8	25,02178975	25	15	0,1	0,003	09/05/2016 07:38:11,582 p.m.
9	25,00196721	25	15	0,1	0,003	09/05/2016 07:38:11,682 p.m.
10	25,00411238	25	15	0,1	0,003	09/05/2016 07:38:11,782 p.m.
11	25,00500049	25	15	0,1	0,003	09/05/2016 07:38:11,882 p.m.
12	25,00498044	25	15	0,1	0,003	09/05/2016 07:38:11,982 p.m.
13	25,00513162	25	15	0,1	0,003	09/05/2016 07:38:12,082 p.m.
14	24,99615239	25	15	0,1	0,003	09/05/2016 07:38:12,182 p.m.
15	24,99450917	25	15	0,1	0,003	09/05/2016 07:38:12,282 p.m.
16	24,99971195	25	15	0,1	0,003	09/05/2016 07:38:12,382 p.m.
17	25,00356568	25	15	0,1	0,003	09/05/2016 07:38:12,482 p.m.
18	25,01502644	25	15	0,1	0,003	09/05/2016 07:38:12,582 p.m.
19	24,99534026	25	15	0,1	0,003	09/05/2016 07:38:12,682 p.m.
20	24,99882089	25	15	0,1	0,003	09/05/2016 07:38:12,782 p.m.
21	25,00053857	25	15	0,1	0,003	09/05/2016 07:38:12,882 p.m.
22	25,0012987	25	15	0,1	0,003	09/05/2016 07:38:12,982 p.m.
23	24,99884283	25	15	0,1	0,003	09/05/2016 07:38:13,082 p.m.
24	24,9911847	25	15	0,1	0,003	09/05/2016 07:38:13,182 p.m.
25	25,00984375	25,41832669	15	0,1	0,003	09/05/2016 07:38:13,282 p.m.

Figure 2.5

Part 2: Add CVT

Sometimes you will need to share tags with code that might run asynchronously in parallel with the DCAF engine. Current Value Table (CVT) is a component that provides a simple interface between DCAF and other LabVIEW code.
During this part of the exercise we will publish the Temperature tag and visualize it in the Front Panel of your TCRL Host Main.vi.

1. Open the **Temperature Controller** project if not already opened.
2. Open the **Configuration Editor** and load **SimulatedSystem.pcfg** if not already opened.
3. Add a CVT module to the Simulation Engine in the same way you added the TDMS datalogger.
4. Select the **CVT** module. Select **To CVT** direction.

Direction ⚪ From CVT ⚫ To CVT
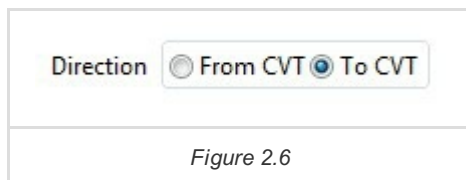
Figure 2.6

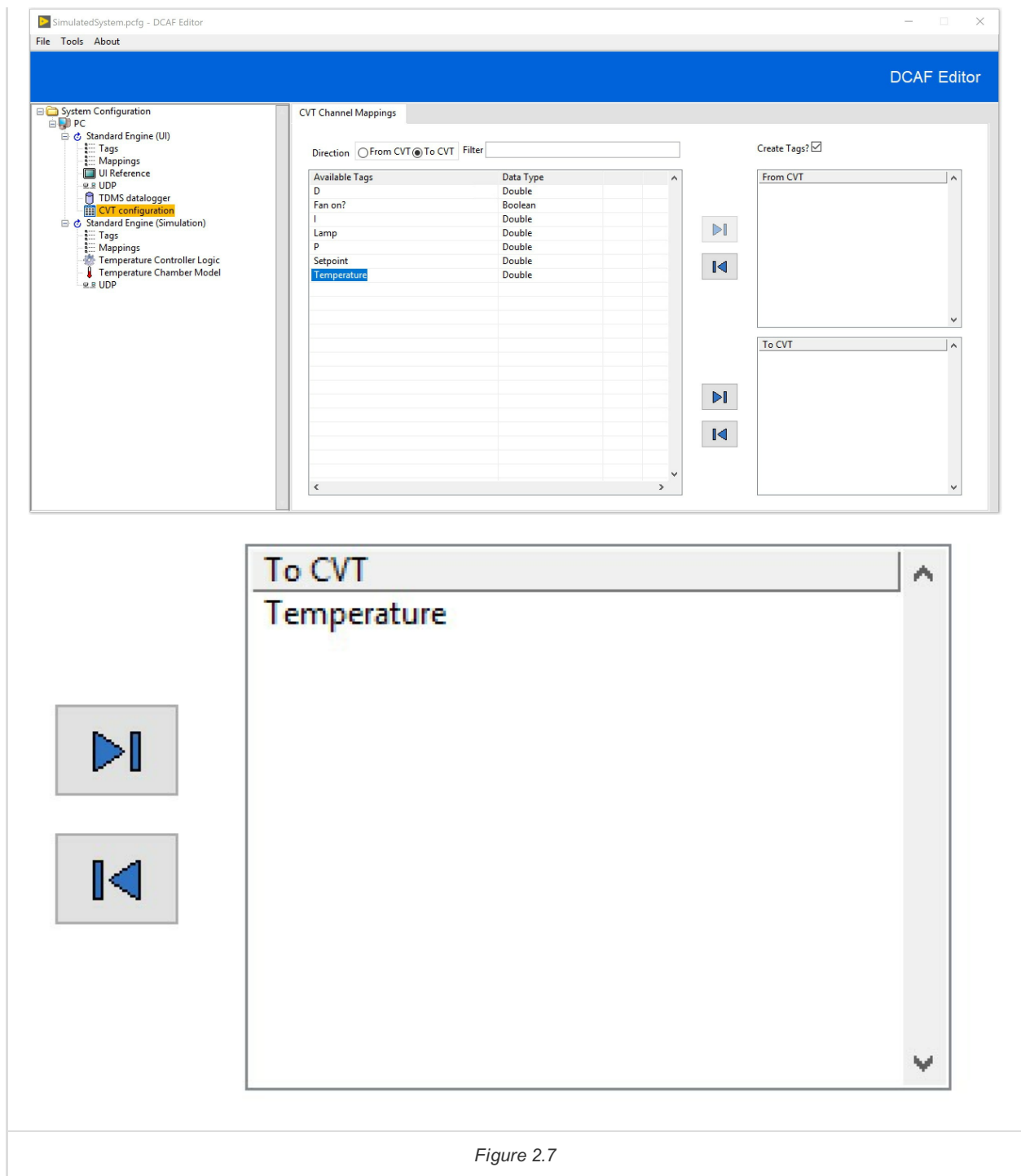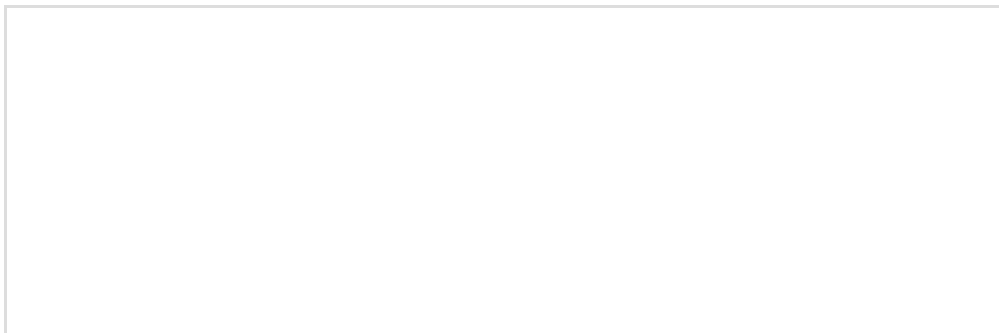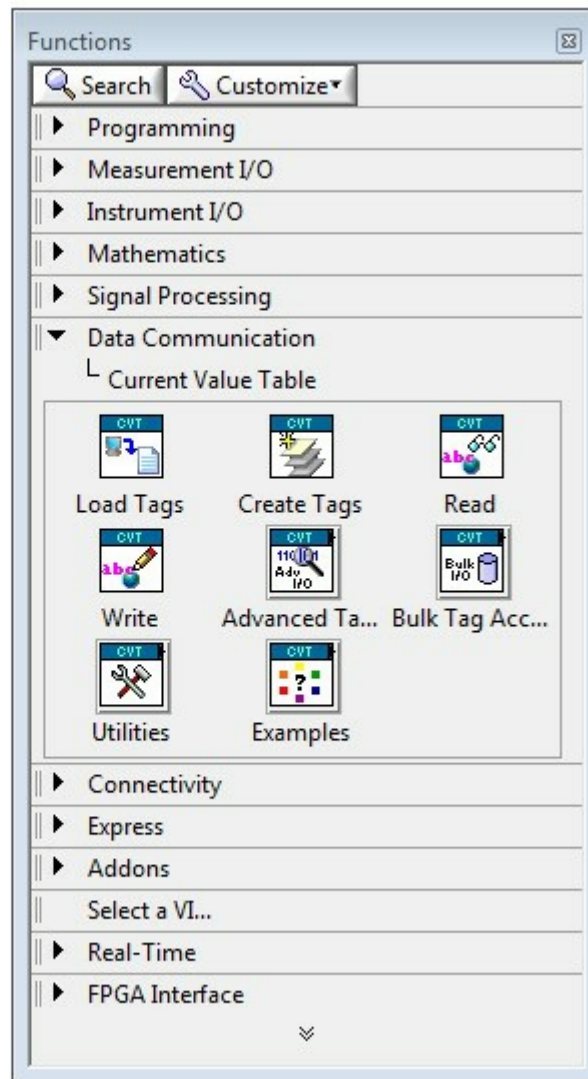5. Move Temperature to the To CVT box.

Figure 2.7

6. Save your configuration.
7. 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 2.8

8. Finish the code as shown in Figure 2.9

Figure 2.9

9. Rearrange the front panel Indicators such that the new **Temperature** indicator is visible as shown in Figure 2.8.
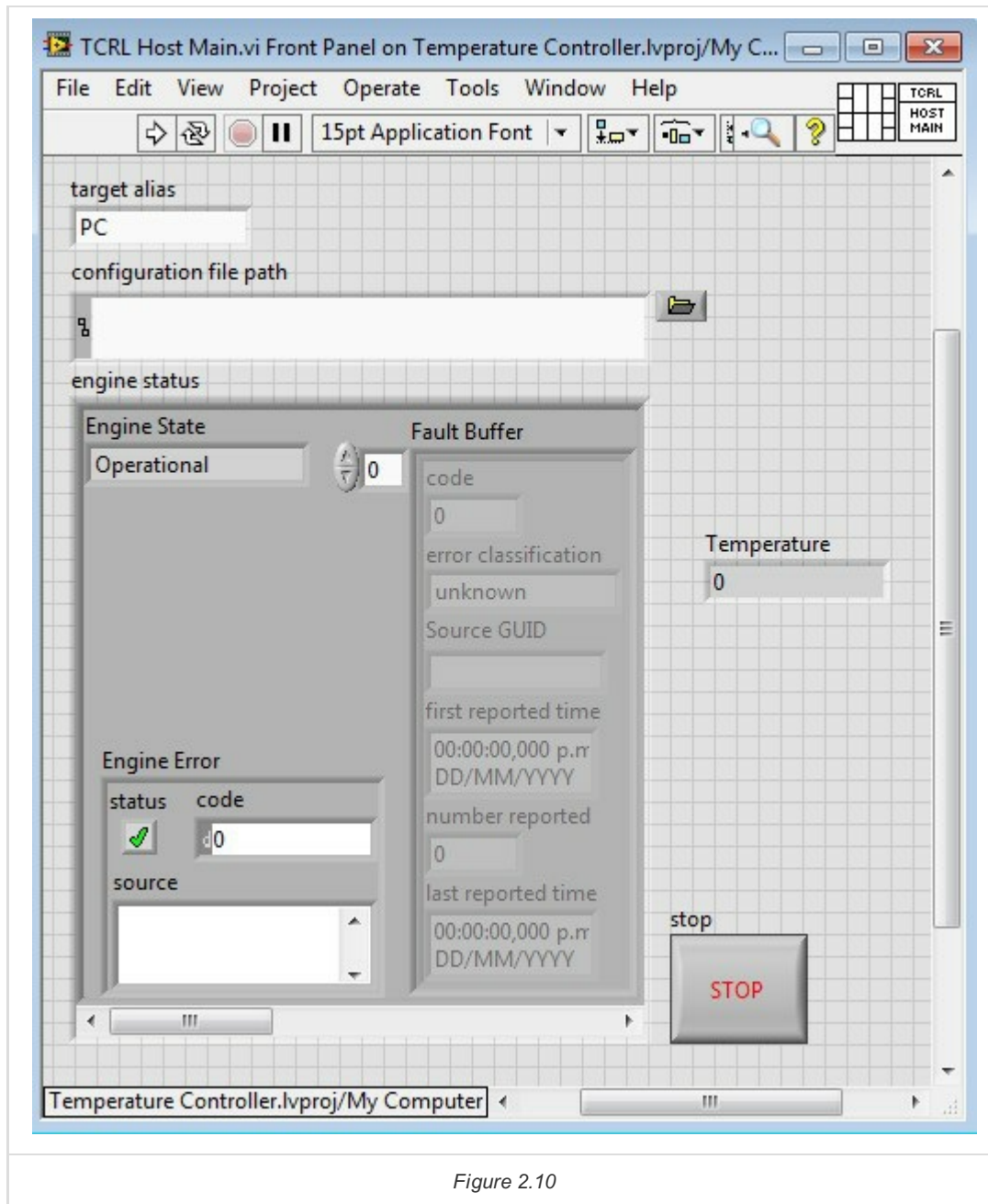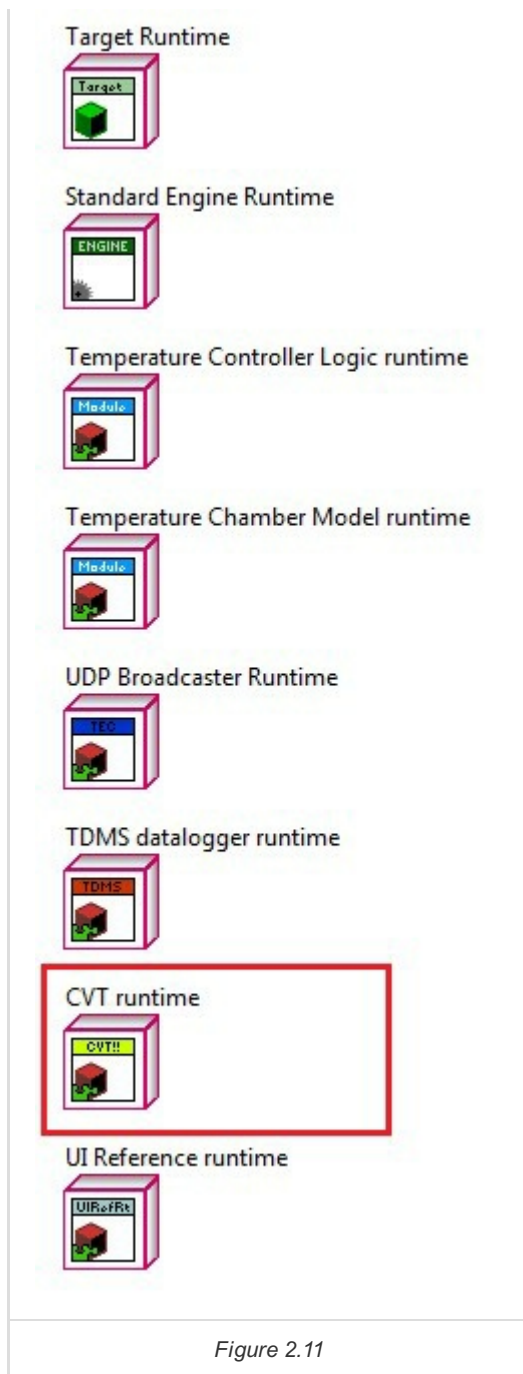

Figure 2.10

10. 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.**

Target Runtime

Standard Engine Runtime

Temperature Controller Logic runtime

Temperature Chamber Model runtime

UDP Broadcaster Runtime

TDMS datalogger runtime

CVT runtime

UI Reference runtime

Figure 2.11

11. Run **TCRL Host Main.vi**. The UI should still be w orking. Verify the value displayed in the new Temperature indicator in **TCRL Host Main.vi** corresponds to the value displayed in the **Temperature** chart in **TCRL User Interface.vi**
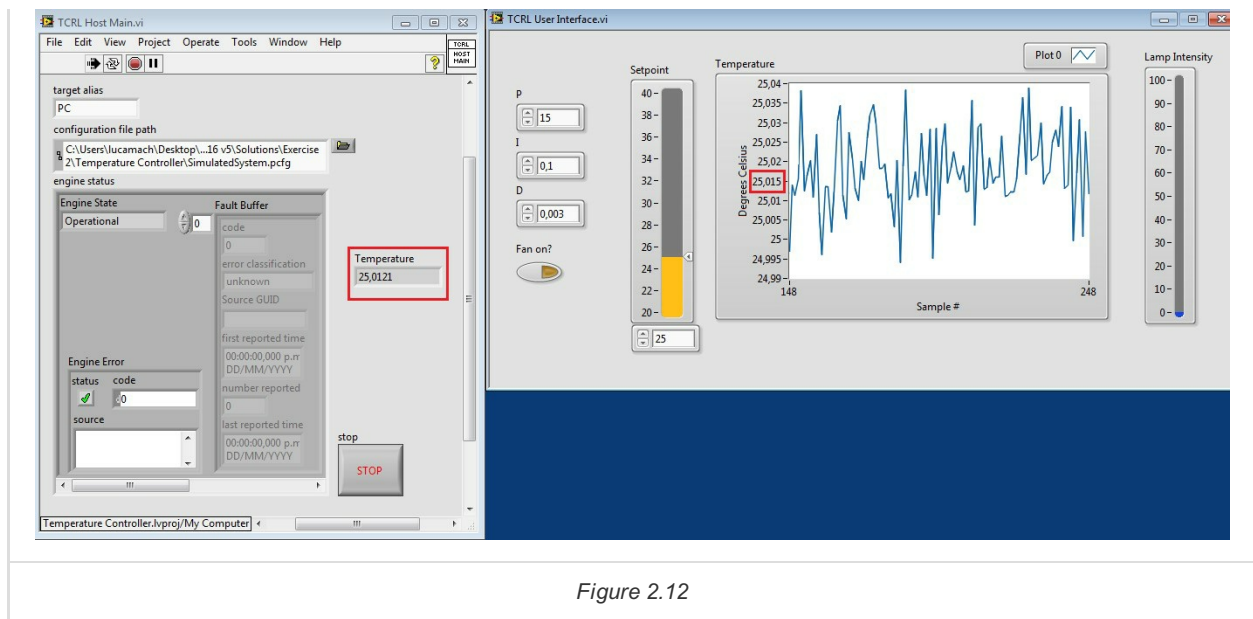
Figure 2.12

12. Stop and close **TCRL Host Main.vi**.
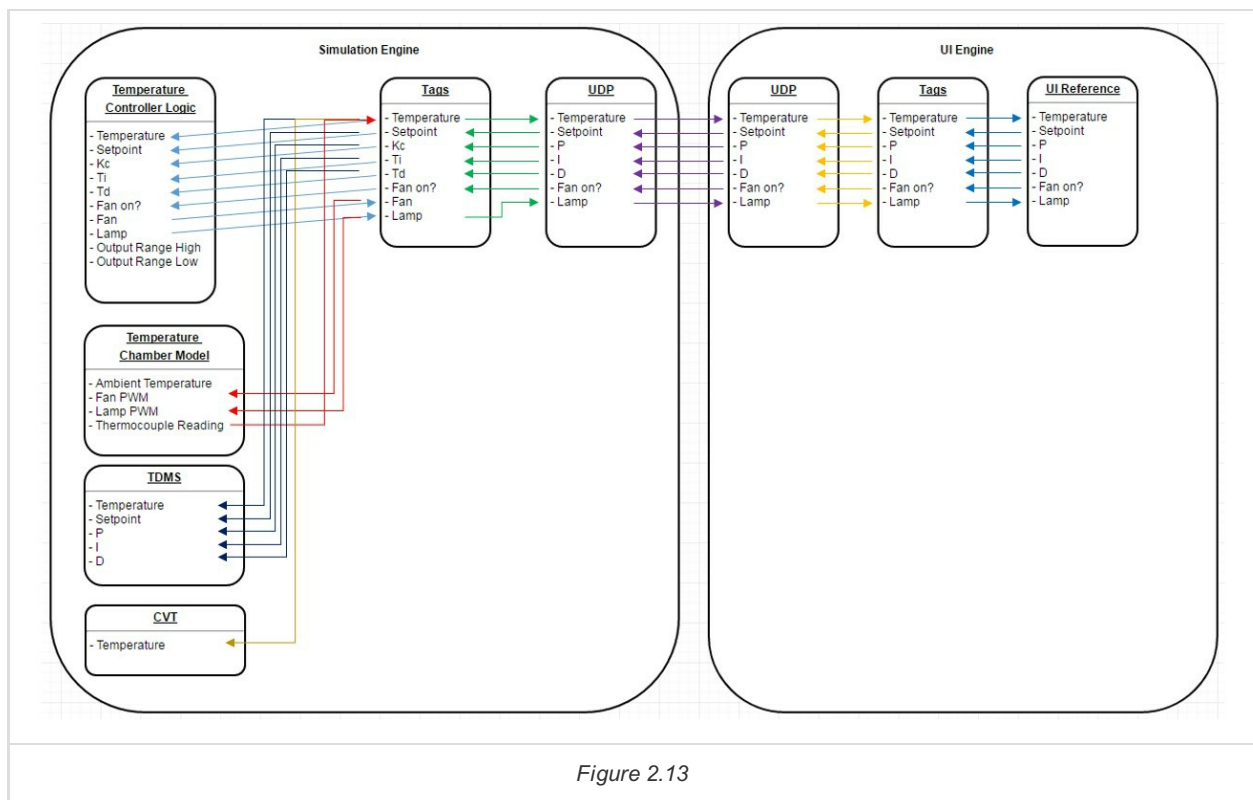13. Take a look to the following diagram to verify the updated mappings.

Figure 2.13