User Defined Function Blocks Lab

Objectives

IN THIS LAB YOU WILL:

• Be able to describe a basic overview of User Defined Function Blocks (UDFB's)
• Write logic using User Defined Function Blocks
• Download the logic to a PACSystems controller
• Interact with the logic using the QuickPanel display
Overview

User Defined Function Blocks (UDFBs) are logic blocks that have parameters and instance (member variables) data. This block type allows users to define their own function blocks, instead of being limited to the standard function blocks provided in the PACSystems instruction set. In many cases, the use of UDFBs results in a reduction in total program size.

Once a UDFB is defined, multiple instances of it can be created and executed. Each instance has its own unique copy of the UDFB instance data, which consists of the UDFB internal member variables and all of its input and output parameters. When a UDFB is called for a given instance, the UDFB logic operates on that instance’s copy of data. The values of the instance data are retained from one execution of the UDFB to the next.

An example of a User Defined Function Block call is shown below. In this example, the UDFB named ACME CONVEYOR is called with the Instance Variable named Conv_1.

UDFB logic can be written in the LD, FBD, or ST programming languages, and can call other types of blocks, even other UDFBs.

The following examples depict an instance of the UDFB MyFB being called. It has two inputs and two outputs. The instance data is the variable ABC.

![Figure 5-1. Example of User Defined Function Block Call](image)

<table>
<thead>
<tr>
<th>In LD logic</th>
<th>In FBD logic</th>
<th>In ST logic</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MYFB</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>MInt</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>MInt</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>MReal</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>ABC</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>ABC</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note: Parameter names of UDFBs and specially function blocks are truncated so they do not take more than half the width of the instance graph. To see a parameter’s full name and other properties in a tool tip, hover the mouse pointer over the parameter. To widen the instance graphic, increase the cell width.

![Figure 5-2. Example of User Defined Function Block Calls in LD, FBD, and ST](image)
User Defined Function Block Facts

The main details about UDFBs are summarized below:

- UDFBs are the same as Parameterized Blocks, except that they have Member Variables – and – they require an Instance variable for the data storage for these Member Variables.
- Each UDFBs instance operates independently from each other instance.
- A unique **Instance Name** is provided for accessing each instance, such as: `UDFB_Name.Instance_Name`
- Each UDFB instance has its own set of unique **Member Variables**. These are visible in the Variable List Navigator in the form of a data structure, as shown for the instance named `ACME_Conveyor.Conv_1` below:

![Figure 5-3. Example of UDFB Instance Data Structure](image)

- UDFB Instances need to be called in order to execute.

**Input Parameters**

UDFB Input Parameters are Read Only inside the UDFB unless they are defined as **Pass by Reference** or **Pass by Value Result**. Input Parameters may also have Initial Values assigned to them.

**Output Parameters**

UDFB Output Parameters are Read Only inside the UDFB and may also have Initial Values assigned to them.
Member Variables

UDFB Member Variables (or Instance Variables) are defined by the user and comprise the internal data structure that is used by the UDFB. Each Member Variable has the following attributes:

- Type
- Length
- Pass By Mechanism
- Retentive
- Initial Value
- Description

How Parameter Data Moves During UDFB Execution

The parameter Pass By mechanism determines how the parameter information is moved into and out of the UDFB. The Pass By attribute may have the values: **By Reference**, **By Value**, and **By Value Result**.

Parameters may be Mapped Variables, Symbolic Variables, and Constants.

Pass By Value

If a parameter is defined as **Pass By Value**, then the parameter data is passed into the UDFB as follows:

- When the UDFB is called, the controller copies the parameter data values into the UDFB
- Any write to that data inside the UDFB does not affect the “real” input that exists outside the UDFB, only the internal copy
- If a Pass By Value parameter is not provided on a Call to the UDFB, the UDFB uses the last data for that parameter

Pass By Value is demonstrated in the following:

![Figure 5–4. Operation of Pass By Value Parameters](image)

101010

MyData

101010

101010
Pass By Value Result
If a parameter is defined as **Pass By Value Result**, then the parameter data is passed into the UDFB as follows:

- When the UDFB is called, the controller copies the parameter data values into the UDFB
- Any write to that data inside the UDFB does not affect the “real” input that exists outside the UDFB, only the internal copy
- When the UDFB exits, it copies the data back to the passing parameter

Pass By Value Result is demonstrated in the following diagram:

![Figure 5–5. Operation of Pass By Value Result](image)

Pass By Reference
If a parameter is defined as **Pass By Reference**, then the address of the parameter is passed into the UDFB such that read and writes to the parameter access the actual Variable that was passed in. Pass By Reference is demonstrated in the following:

![Figure 5–6. Operation of Pass By Reference](image)

Creating UDFB Instances
You create an instance of a UDFB by calling it in your logic and assigning an **Instance Name** in the function properties (????).

![Figure 5–7. UDFB Instance Call without Instance Name](image)
In the following example, the UDFB **ACME_Conveyor** is called. The variable associated with the instance is **Conv_R45**.

![Diagram](image)

**Figure 5–8. UDFB Instance Call with Instance Name**
Lab Exercise – Building Logic with UDFB

In this exercise you will be using a UDFB that has already been created to build some logic. This lab assumes the following basic hardware configuration and setup. The IP Addresses defined below are used in the Project provided with this lab.

1. Begin by **double-clicking** on the Project named “DiscCafe_UDFB” in the Navigator window.
2. In the **Program Blocks** section of the **PAC_1 target**, double-click on the block called **_MAIN** to open it in the editor window.

3. Next, click on the block labeled **Start_Warning** and look at its properties in the **Inspector** window.
4. Notice that this block was written in **Ladder Diagram** and it is of the type **Function Block**. Click the mouse in the field to the right of the word **Parameters** in the **Inspector window**. A button will appear in the field.

![Inspector window](image)

<table>
<thead>
<tr>
<th>Block Properties</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>START_WARNING</td>
</tr>
<tr>
<td>Description</td>
<td>Provides a 10 second warning before motor starts</td>
</tr>
<tr>
<td>Language</td>
<td>Ladder</td>
</tr>
<tr>
<td>Block Type</td>
<td>Function Block</td>
</tr>
<tr>
<td>Parameters</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Lock Settings</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extra Local Words</td>
<td>128</td>
</tr>
<tr>
<td>Extra Discrete Space</td>
<td>32</td>
</tr>
<tr>
<td>Extra Non-Discrete Space</td>
<td>32</td>
</tr>
</tbody>
</table>

5. Click on the button to view the parameters of the UDFB. You will see **2 Input Parameters** labeled **STOP** and **START**.

![Parameters window](image)

<table>
<thead>
<tr>
<th>Input</th>
<th>Name</th>
<th>Type</th>
<th>Length</th>
<th>Pass By</th>
<th>Ref</th>
<th>Init Val</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>STOP</td>
<td>BOOL</td>
<td>1</td>
<td>Value</td>
<td></td>
<td></td>
<td>Stop FB</td>
</tr>
<tr>
<td></td>
<td>START</td>
<td>BOOL</td>
<td>1</td>
<td>Value</td>
<td></td>
<td></td>
<td>Start FB</td>
</tr>
</tbody>
</table>
6. Click on the **Outputs** tab and you will see that the block has **3 Output Parameters** labeled **MOTOR**, **HORN**, and **LIGHT**.

![Output Parameters Table]

7. Click on the **Members** tab and you will see that block has **2 Member Variables** labeled **DELAY** and **CR**.

![Member Variables Table]

The member variables do not have the public property checked so they are only accessible inside the UDFB.

8. Click the **OK** button to close this dialog box.
9. Double-click on the **Start_Warning** block to view its logic.
This UDFB provides a 10-second warning that a motor is about to start. During these 10 seconds, a light blinks on and off and a horn sounds an alarm (or the outputs of the block can be used to energize any 2 devices of the user's choice) after which the motor output turns on.

The pulse timer (TP) labeled Seal_Time turns on its' output for 2 seconds giving the auxiliary contact from the motor starter enough time to close and seal-in the circuit. If the motor starter's auxiliary contact doesn't close within 2 seconds, the motor output turns off.

10. Close the Start_Warning block by clicking on the “x” in the corner of the editor window.

11. In the Navigator window, click on the Start_Warning block and drag-and-drop it to the open _MAIN block in the Editor window.

You now have a copy of the User Defined Function Block called Start_Warning in the _MAIN ladder block.
This block needs to have an instance name assigned to it.

**NOTE**

All of the required variables and data structures have been created for you ahead of time so you will only need to select them from a list.

12. **Double-click** on the **question marks** inside the block. Select **_MAIN.Conveyor_1** from the list.
13. Next, select the **Horizontal/Vertical Wire** tool from the instruction set **toolbar**.

![Horizontal/Vertical Wire tool](image)

14. Use this tool to connect the block directly to the left rail of the ladder.

![Block connection](image)

15. Now, **double-click** in the **cell** next to the **STOP input** of the block and select **C1_Stop_PB** from the list.
16. Select the remaining variables as shown below.

**NOTE**

Because we do not have an actual auxiliary contact from a motor starter, use C1_Motor as the AUX input for this exercise.
17. Drag-and-drop 2 more **Start_Warning** blocks into the _MAIN_ ladder block and configure them as shown below.

18. When you have finished, **right-click** on the target labeled **PAC_1** and select **Validate** from the target right mouse menu. This will check your work for errors.

19. If no errors are present, **right-click** on the **PAC_1** target and select **Go Online** from the menu, or press the **Online/Offline** toolbar button 🌡️.
20. Next, right-click and select the **Online Commands, Set Programmer Mode** menu, or press the **Toggle Online Mode** toolbar button.

21. Next, select the **Online Commands, Stop PLC** menu, or press the **Stop Active Target** toolbar button.
22. Click **OK** to stop with outputs disabled.

23. **Right-click** and select the **Download to PLC** menu, or press the **F8** function key, or press the **Download Active Target** toolbar button.
24. Make sure that the items in the dialog box are checked as shown and then click **OK**.

25. Select the **Online Commands**, **Start PLC** menu, or press the **Start Active Target** toolbar button ▶.
26. Click the **OK** button to start with Outputs Enabled.

27. Check the information displayed on the Status Bar to verify that the PACSystems controller is in Run Mode, and that the Configuration and Logic are both Equal.

Once you have successfully downloaded and started the controller, you can use the QuickPanel to exercise the application. But first, you need to download the QuickPanel application to the demo unit.
28. **Right-click** on the **PAC_1** target and select **Go Offline** from the menu.

29. **Right-click** on the **QP_1** target and select **Set as Active Target** from the menu.

30. **Right-click** on the **QP_1** target and select **Download and Start** from the menu, or press the **F9** function key, or press the **Download and Start Active Target** toolbar button.
31. After the QP_1 application has been successfully downloaded to the QuickPanel go to the QuickPanel screen on the PACSystems demo unit and continue as instructed below.

32. Now use the QuickPanel screen to Start and Stop the motors. Notice that each instance of the UDFB is independent and controls its' own motor.

If you want to examine the execution of the UDFB logic while you interact with the QuickPanel, you can go back online with the controller.
33. **Right-click** on the PAC_1 target and select *Set as Active Target* from the menu.

34. **Right-click** on the PAC_1 target and select *Go Online* from the menu, or press the **Online/Offline** toolbar button.

35. Next, **right-click** and select the *Online Commands, Set Programmer Mode* menu, or press the **Toggle Online Mode** toolbar button.
36. Go to the _MAIN logic block, and **Double-click** on the first **Start_Warning** UDFB with the Instance Name **Conveyor_1**. This will open the **Start_Warning** UDFB with the context of the calling block. You can now examine the parameters that have been passed to the UDFB.
The following shows the contents of the **Start.Warning** UDFB with values relative to the **Conveyor_1** calling block.
Review

In this lab, you have been shown how:

- A User Defined Function Block (UDFB) is configured.
- Logic is created using a UDFB.
- Each UDFB instance is independent.
- To monitor the execution of a UDFB.