OCS Training Workshop
LAB16

IEC Languages

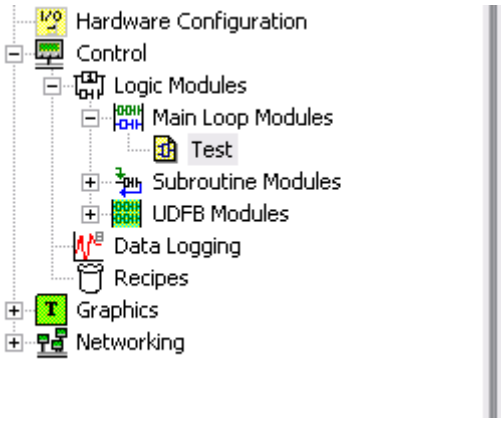
Instantiated versus Non-instantiated Data
(Structures/UDFBs and Subroutines).

Introduction

The training module will introduce difference between instantiated and non-instantiated data in the OCS.

Working with structures

Create a new IEC Program for the OCS with for now a single main loop FBD function block ‘Test’.



In the program variable list right click and select the ‘Add Structure’ option. Give the name of the structure as ‘TestStructure’

Now select the structure and add a variable. There are two options for doing this – select it with a left click in the variable window and press the ‘Ins’ key, or right click on it in the variable window and select ‘Add Variable’

Add the following member variables to the structure: BoolVar of type BOOL; IntVar of type INT; RealVar of type REAL

The structure definition should now appear as follows in the variable window:

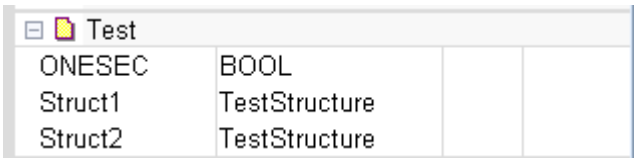
TestStructure	
BoolVar	BOOL
IntVar	INT
RealVar	REAL

In the Test module data create a variable ONESEC of type BOOL with tag ‘S5’. %S5 is the one second pulse bit.

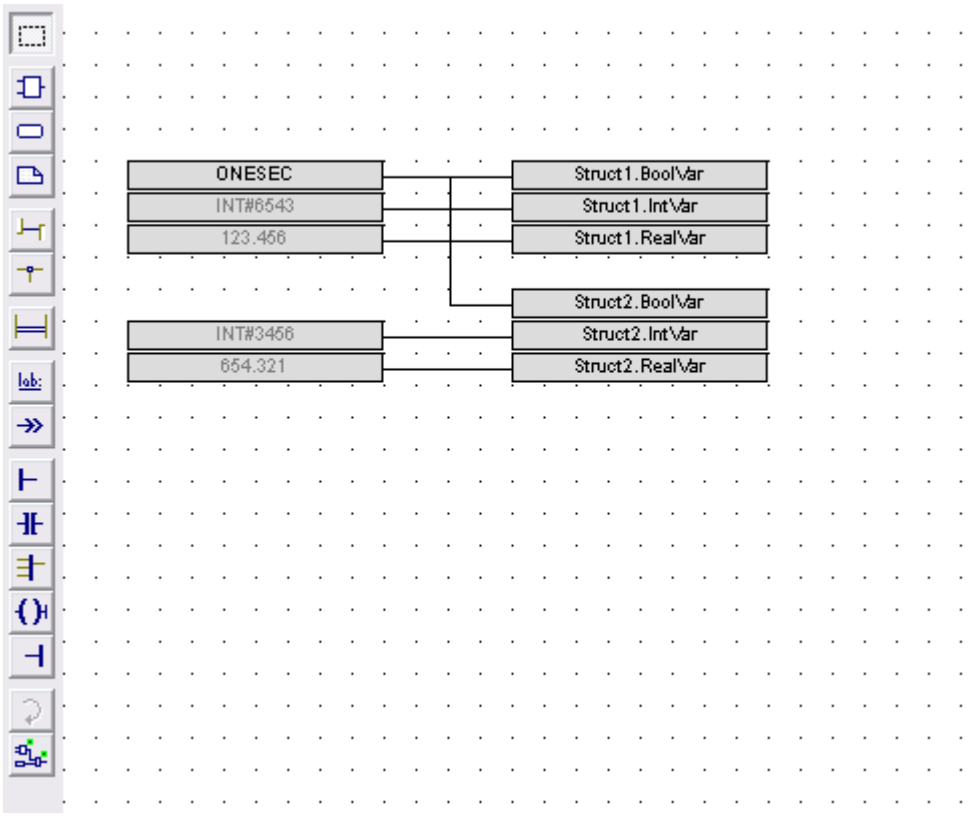
Now click on the TestStructure declaration in the project variable window and drag it in to the data for module ‘Test’

Name the added structure as ‘Struct1’, then repeat adding the structure – this time as ‘Struct2’

The variables for module ‘Test should appear as follows.

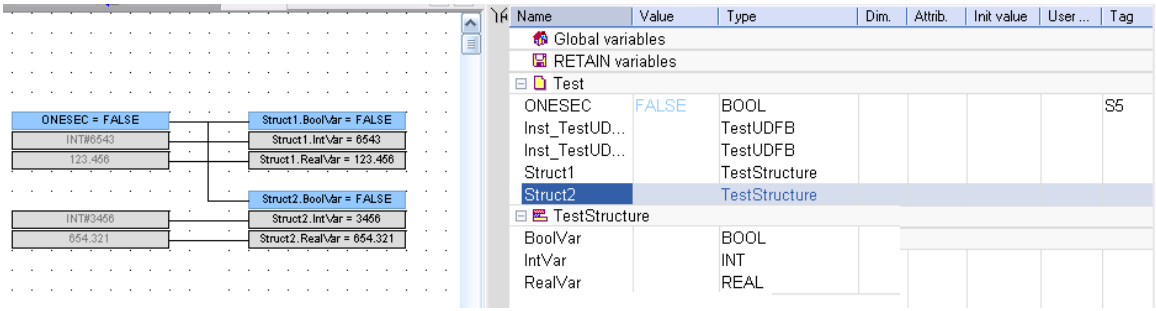


Create the following FBD as ‘Test’



Create one lamp on the first screen and map it to ONESEC. Then compile and download the project to the OCS.

Make sure the OCS is in run mode, and the go to debug mode.
The screen should initially appear as follows:



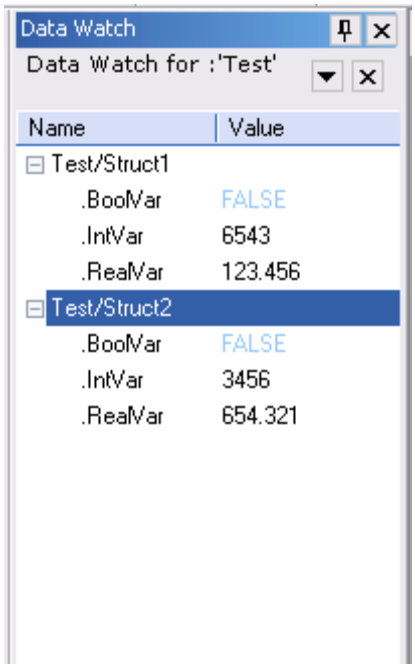
Note the following:

Live structure is displayed in the logic page.

Struct1 and Struct2 cannot be expanded to element level to view the data in them.

Data does not appear in the TestStructure definition because we cannot know which structure we wish to display. (Even if only one structure of the type has been created)

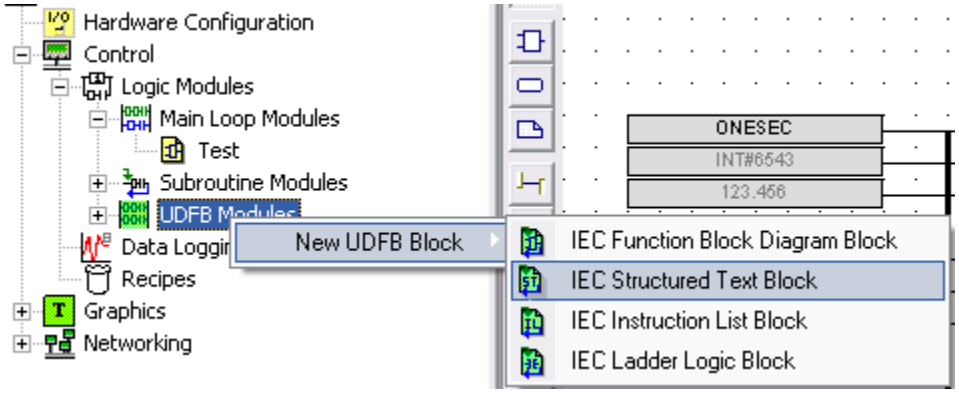
To display the structure data open Data Watch list window and drag the structures from the variable list into the Data Watch and expand the elements.



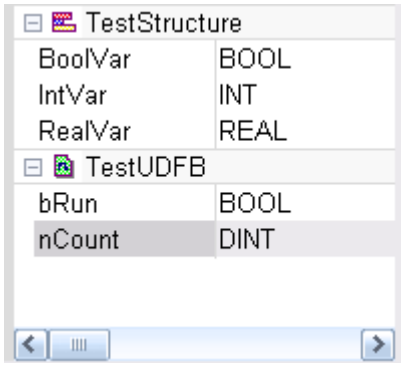
UDFBs

User defined function blocks can be considered as an extension of structures (For those of you with object oriented programming experience they can be considered as an object with a single update method.)

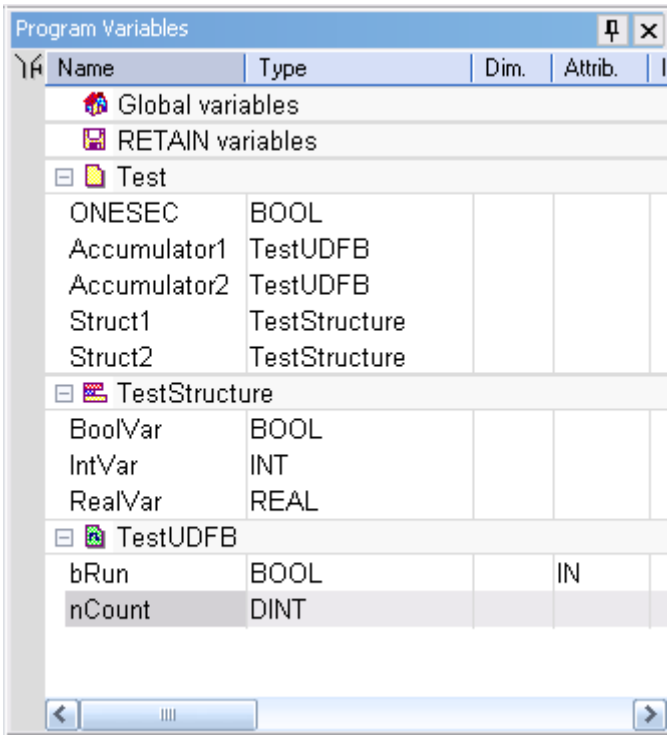
First let's create a UDFB – this time we'll write the module in Structured Text. In our existing Structures program right click in the project navigator window on the UDFB node, select IEC Structured Text Block and name it as TestUFDB.



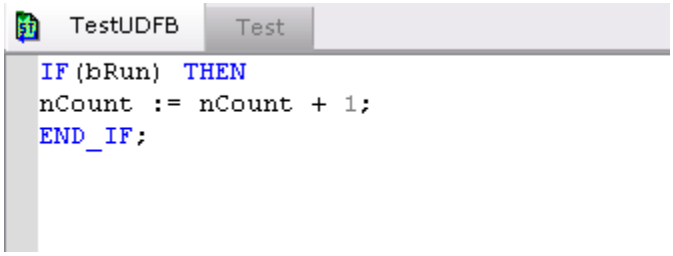
In a similar way to adding member variables to the structure add two variables to the UDFB: bRun as a BOOL Input variable; nCount as a DINT private variable.



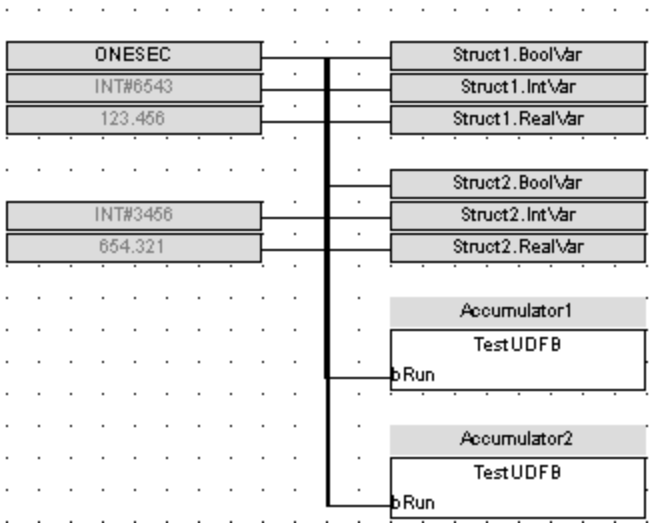
Again add some UDFB instances to the variable list for the main loop by dragging and dropping them into the Test variable folder. Name them as Accumulator1 and Accumulator2.



Now add the control logic to the UDFB definition.

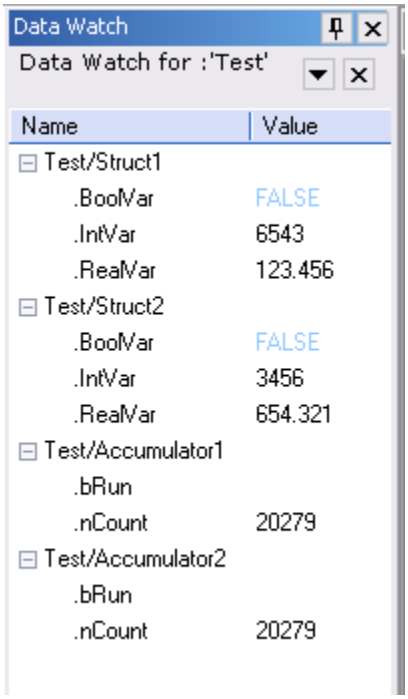


And add the UDFB logic to the main block.



Download and run the application.

Note again that to view the values for nCount for each UDFB the values must be dragged and dropped to the Data Watch window in order to view them.

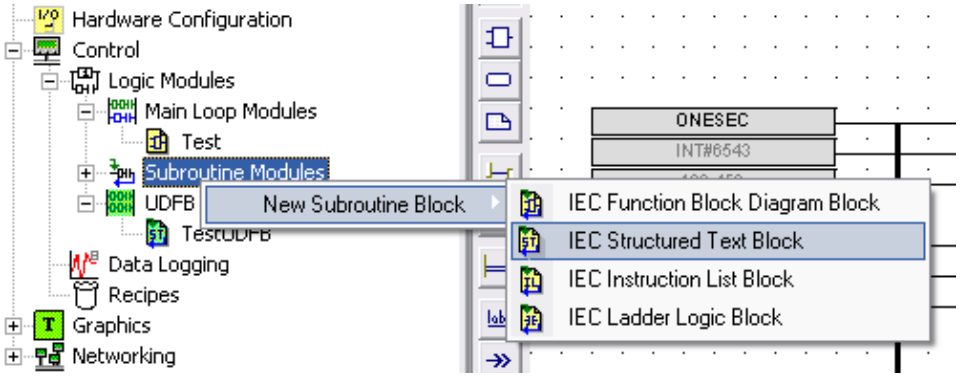


Some items to note:

- Input and output parameters from UDFBs cannot be viewed in the Data Watch
- nCount may be edited independently for each Accumulator block

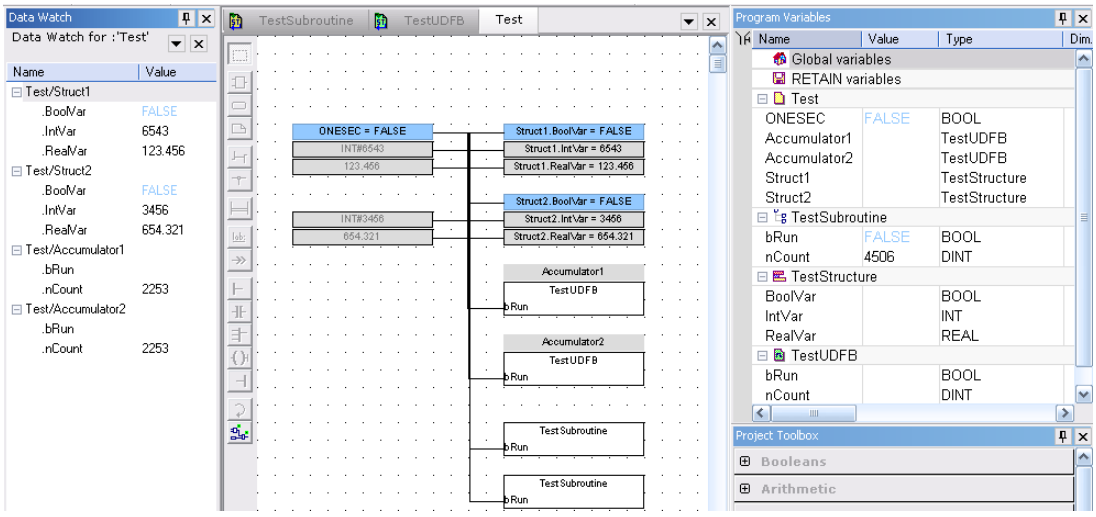
Subroutine Support.

Create and add a Subroutine, again in Structured Text language.



Use the same variables and code as the UDFB for the subroutine and add two calls to this subroutine to the main loop FBD (to place the subroutine on the screen open Project Toolbox -> [Project]).

Download and go to debug mode and bring up the Data Watch..



Some things to note:

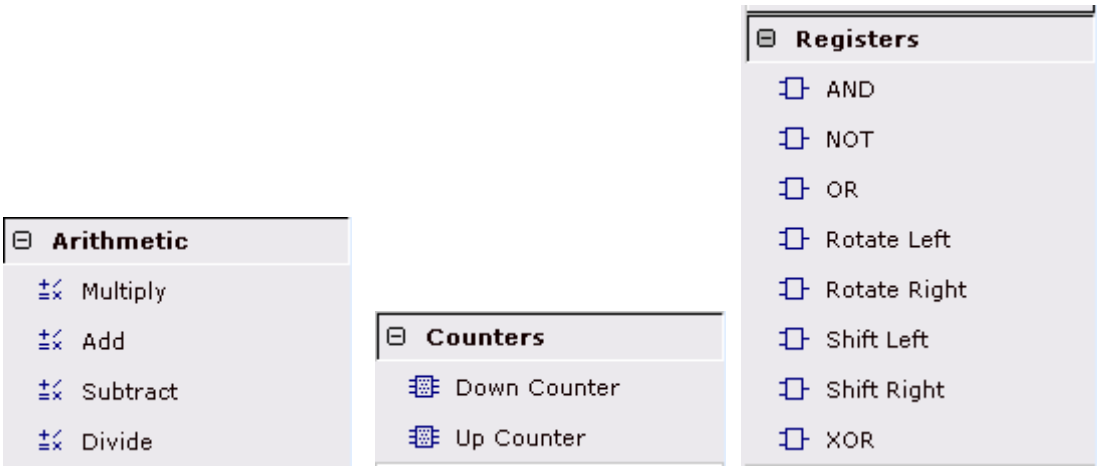
1) The subroutine nCount is double that of the Accumulator nCounts. That is because a subroutine maintains just one set of private data for each subroutine. The two calls to Accumulator1 and Accumulator2 increment the private nCount of each UDFB. The two calls to TestSubroutine each act on the same common private variable. Hence nCount for the TestSubroutine counts up twice as quickly.
2) Modifying nCount for the UDFB instances will not affect the nCount of the other UDFB instance. There is only one non-instantiated nCount for the Subroutine hence modifying it will affect only one data location.

3) Since there is only the single allocation of the storage for private variables of a subroutine these values can and are displayed in the variable list.

Instantiated vs Non-Instantiated standard function blocks

With the instantiated (UDFB) vs non instantiated (Subroutine) blocks in mind consider our function blocks in out standard IEC toolkit.

Consider these expanded groups from our standard toolkit.



Note the difference in symbols between the various operations and the counter operations. The dotted form of the function block indicates that the function must be instantiated – it has it's own private workspace.

The other two block styles do not require instantiation – they require no private workspace. The outputs can be generated from the current inputs directly.

A way to consider it is that if a block needs to maintain what has happened previously to operate then it will need to be instantiated. In this case the counter requires the previous count in order to track the current count. Other good examples of blocks requiring instantiation are timer blocks and PID blocks.

To include a block not requiring instantiation in your logic it can be dragged directly from the toolbox in to your logic page or script and then its inputs and outputs configured.

To include a block which needs to be instantiated in your program first of all you must instantiate it by dragging it from the toolbox in to the variable list for the appropriate program module (or retain/global section), then dragging the instantiated variable from the variable list to the logic page or script in which it is to be used.

End of LAB 16