

Using an EXOR HMI with an SMLC

- This tutorial covers connecting an EXOR HMI to an SMLC via Ethernet.
- Communications will use the standard Modbus/TCP protocol supported by both the EXOR HMI and the SMLC.
- This tutorial covers an Ethernet connection between the two devices so the optional Ethernet port on the EXOR is required.
- This tutorial assumes basic familiarity with SMLC IEC 61131-3 programming.



Ethernet (Modbus/TCP) connection

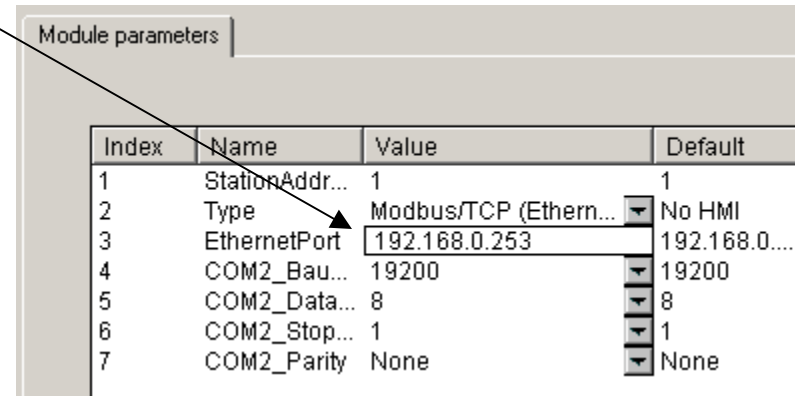
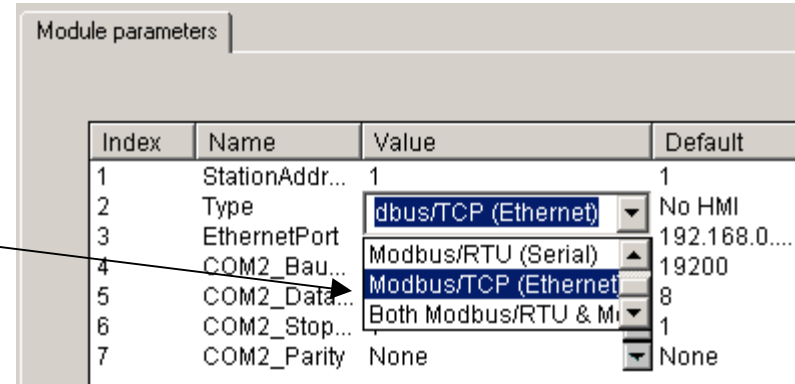


SMLC to EXOR HMI Tutorial - Basic concepts

- This tutorial will cover the transfer of the following data types between an SMLC and an EXOR HMI. The IEC-61131-3 data type is in parentheses.
 - Boolean, single bit data (BOOL)
 - Integer, 16-bit data (INT)
 - Double Integer, 32-bit data (DINT)
 - Floating point, 32-bit data (REAL)
 - Strings, 40 characters max (STRING)
- The SMLC contains a bank of 2000 “virtual” Modbus registers. These registers simulate “holding” registers in a Modicon PLC (4xxxx).
- All access from the EXOR to the SMLC will be via these “holding” registers.
- These virtual Modbus registers can be mapped to IEC addresses in blocks of 40 at a time.
- Each block of registers can be mapped to either:
 - 40 INTs
 - 20 DINTs
 - 20 REALs
 - 1 STRING (80 characters)
- The IEC address corresponds to the Modbus register number. E.g. %QW1 = Register 400001 = Register 1, %QW2 = Register 400002 = Register 2, etc.
- The HMI cannot directly access the SMLC physical I/O. If you need this functionality your program will have to copy physical I/O points to HMI variables and vice versa.
- HMI variables are initialized to 0 at SMLC powerup so if you want HMI values to be retain/persistent you need to copy the retain/persistent value to the HMI variables once after powerup and then copy the HMI values back to the retain/persistent variables every scan after that.

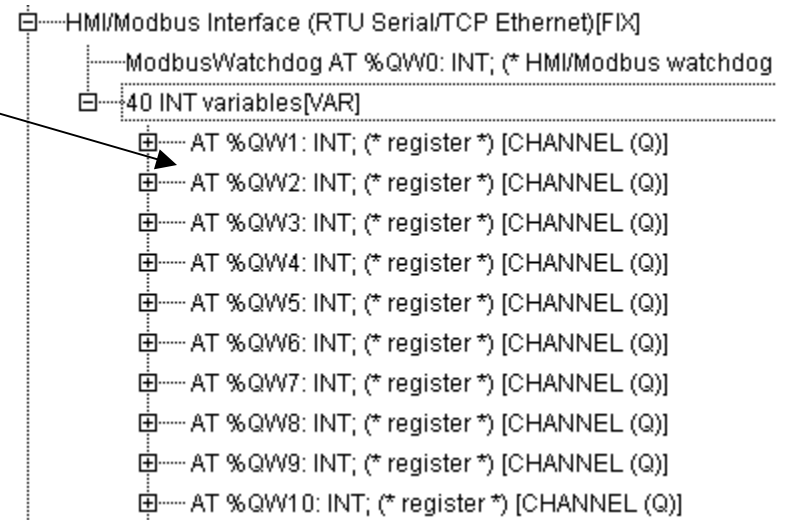
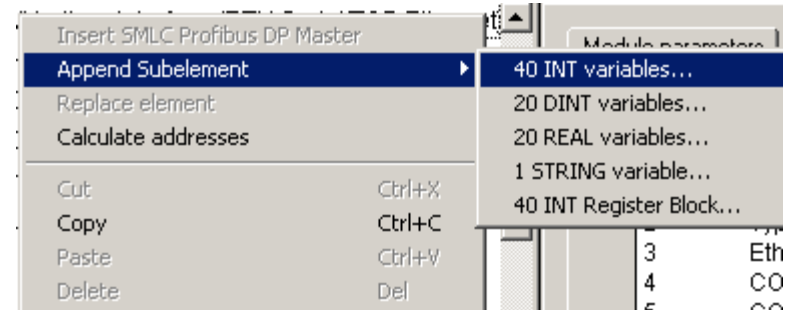
SMLC Configuration - Configuring the HMI driver

- In CoDeSys go to the PLC Configuration section of the Resources tab
- Select the HMI/Modbus Interface section of the SMLC - HMI/IO Configuration tree
- On the Module parameters tab select Modbus/TCP (Ethernet)
- Enter the IP Address of the SMLC Ethernet port that you will be connecting the HMI to. In this example we're going to connect the HMI to the EN0 port on the SMLC which is at the default IP address of 192.168.0.253.



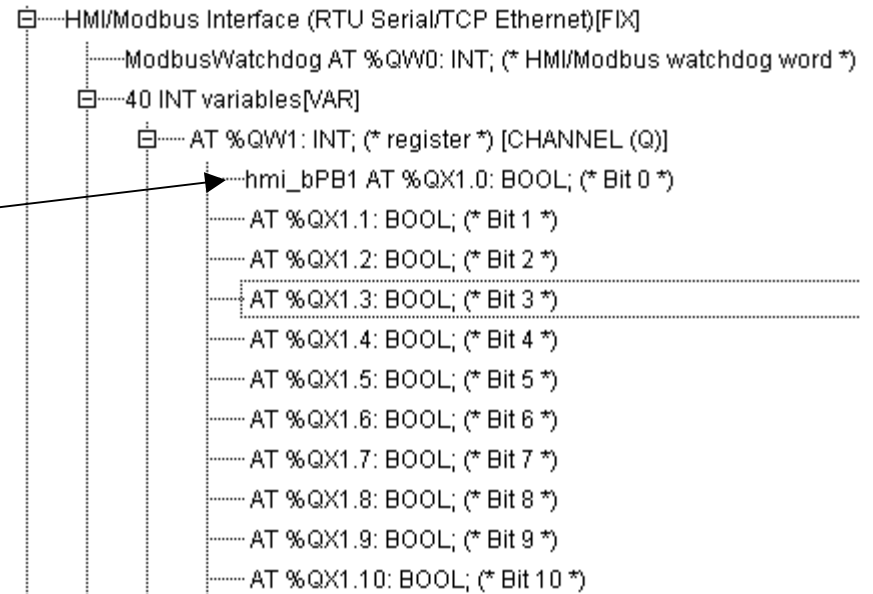
SMLC Configuration - Creating the Modbus Registers

- Right click on the HMI/Modbus Interface in the PLC Configuration tree and select Append Subelement | 40 INT variables.
- This adds the first bank of 40 registers.
- The IEC address of the register corresponds to the Modbus register number. The Exor uses 4xxxxx nomenclature for Modbus registers so
400001 = %QW1
400002 = %QW2
400003 = %QW3
etc.



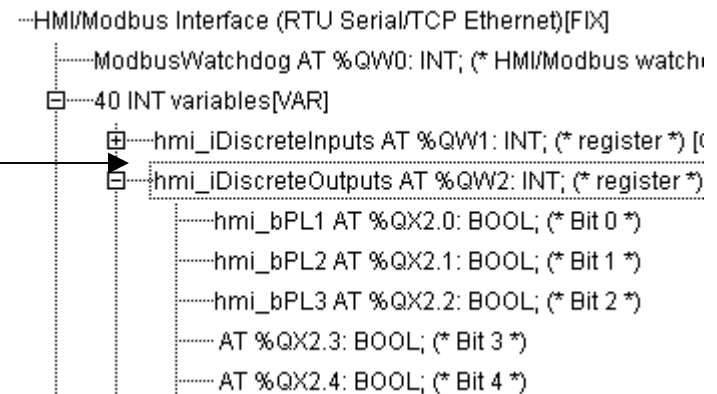
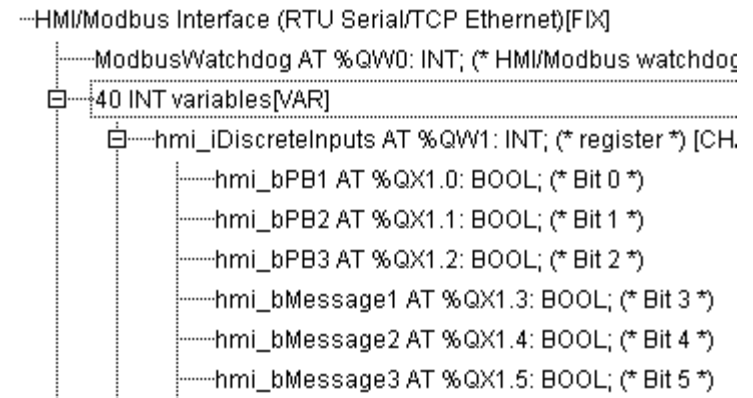
SMLC Configuration - Boolean variables

- In our HMI we want to have some bit level, or boolean devices such as pushbuttons and pilot lights.
- To assign a bit level device we will use the bits of individual registers.
- Expand the register at %QW1 and double-click on the AT to the left of the %QX1.0.
- Enter the variable name to be used in the SMLC program.
- By convention we use a prefix hmi_ to indicate that this is an hmi variable and a b to indicate that its a boolean for a final variable name of hmi_bPB1.



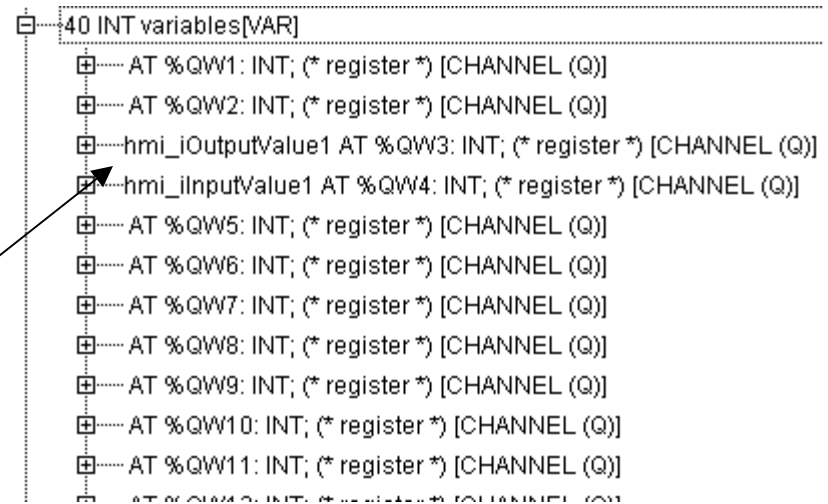
SMLC Configuration - Boolean variables

- Add some more boolean variables for later use in our tutorial.
- We will add a total of 3 pushbutton inputs, 3 inputs for selecting different string messages using a selector switch and 3 pilot light outputs.
- **TIP** when creating boolean variables for the HMI put the input bits and output bits into separate words in the PLC Configuration. If you mix input and output bits in the same word you could have a race condition between the HMI and PLC program as to who updates the its outputs last.
- **TIP** It may be useful to assign variable names to the words that contain the bits. This can help keep things organized. In this example we have named the words hmi_iDiscreteInputs and hmi_iDiscreteOutputs.
- When you're done things should look like this in registers 1 and 2.



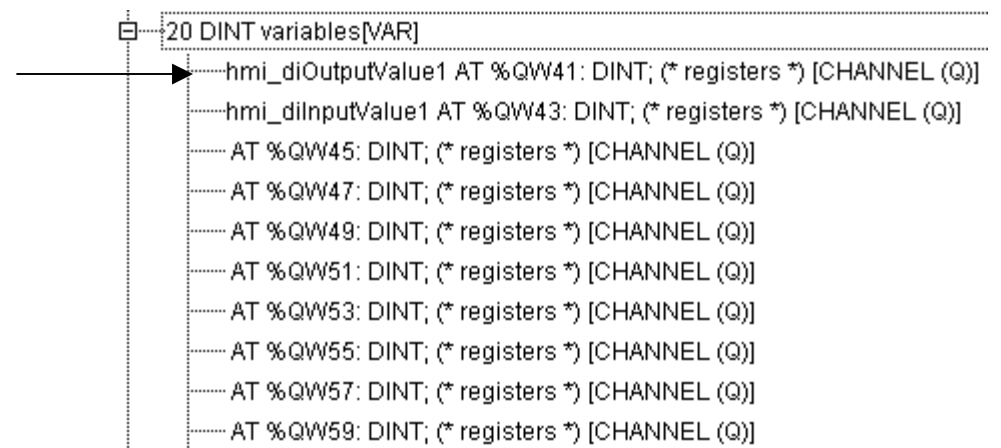
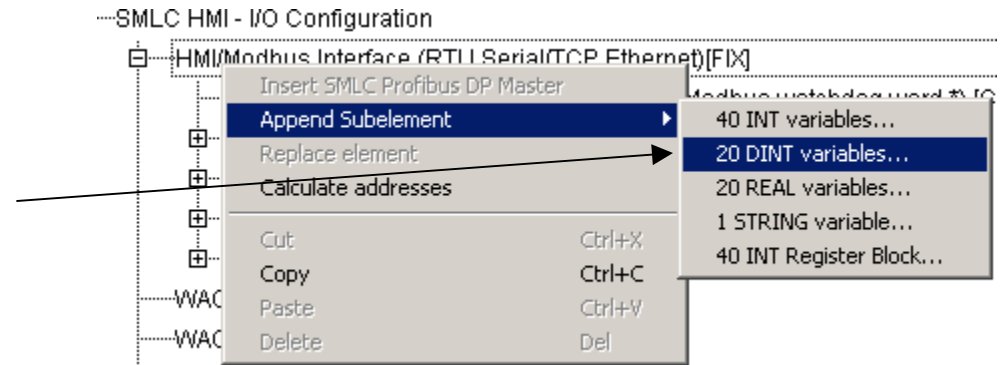
SMLC Configuration - Integer variables

- We want to have some integer level devices for numeric data entry and display.
- To assign a integer variable we will use entire registers.
- Registers in the SMLC are signed integers or INT in IEC type nomenclature.
- The Exor supports Signed and Unsigned word access.
- Double-click on the AT to the left of the %QW3.
- Enter the variable name to be used in the SMLC program.
- By convention we use a prefix hmi_ to indicate that this is an hmi variable and a i to indicate that its an integer for a variable name of hmi_iOutputValue1.
- Add another integer variable at %QW4 called hmi_iInputValue1.



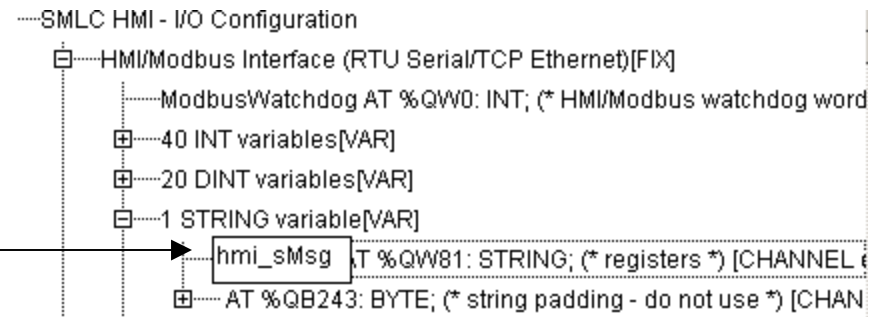
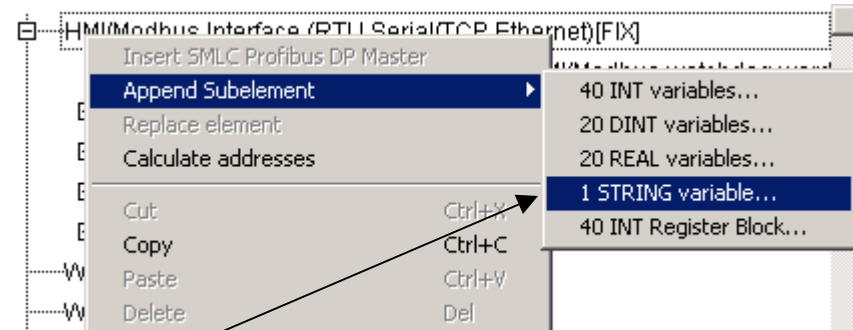
SMLC Configuration - Double Integer variables

- In our HMI we want to have some double integer (32-bit) level devices for displaying or entering very large values.
- Right-click on the HMI/Modbus Interface and append 20 DINT variables.
- Note that DINT register numbers increase by two because each 32-bit value is two 16-bit Modbus registers.
- Double click on the AT in front of %QW41 and enter the variable name to be used in the SMLC program. By convention we use di to indicate that its a DINT for a variable name of hmi_diOutputValue1.
- Add another variable at %QW43 called hmi_dilnputValue1.



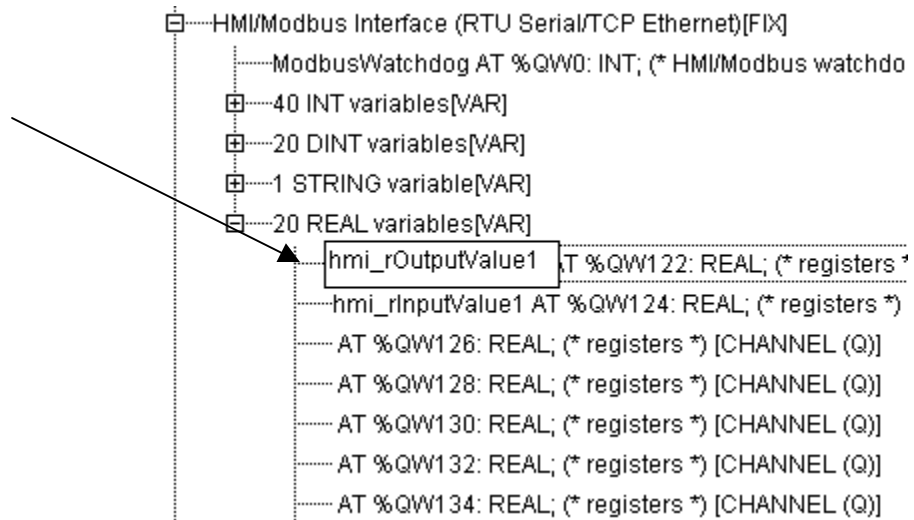
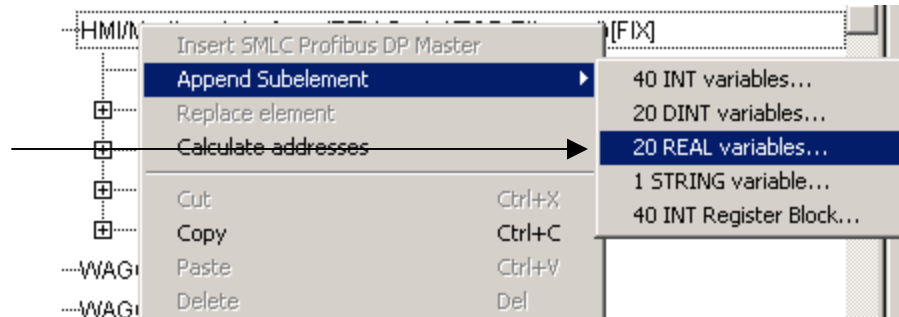
SMLC Configuration - String variables

- In our HMI we want to have a string variable to display messages that are formulated by the SMLC program.
- The EXOR has the capability to store internal message lists that are called up by a value in a register. For that type of message you can simply use an integer register. In this example we will actually build custom strings within the SMLC program and display them in a string variable.
- Right-click on the HMI/Modbus Interface and append 1 STRING variable.
- Each string takes 40 registers and can contain up to 80 characters. Note that in the EXOR strings are limited to 40 characters.
- Double click on the AT in front of %QW41 and enter the variable name to be used in the SMLC program. By convention we use s to indicate that its a STRING for a variable name of hmi_sMsg.



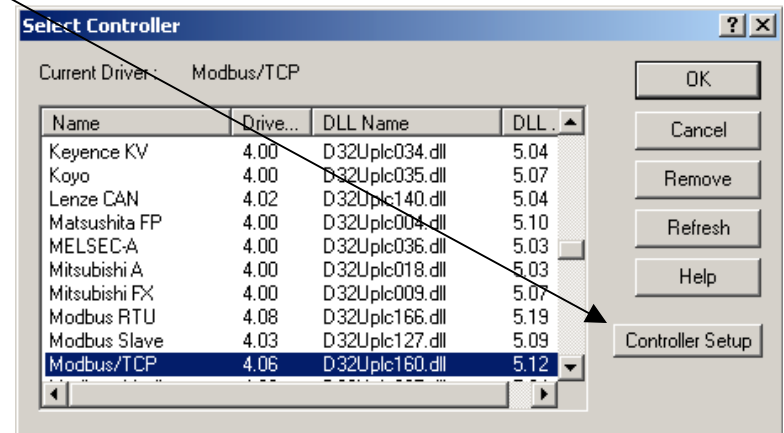
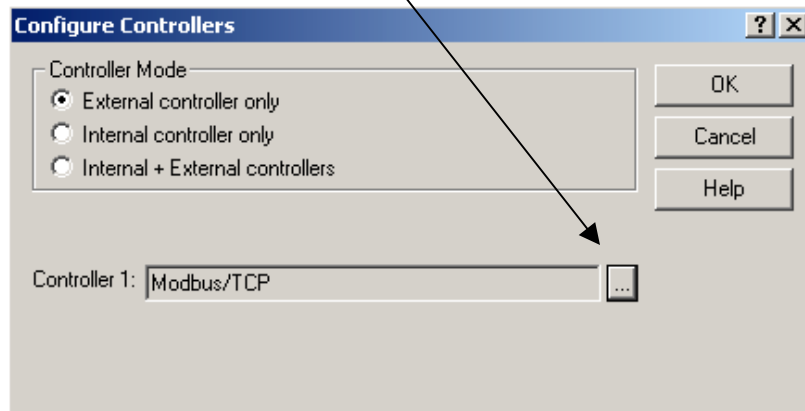
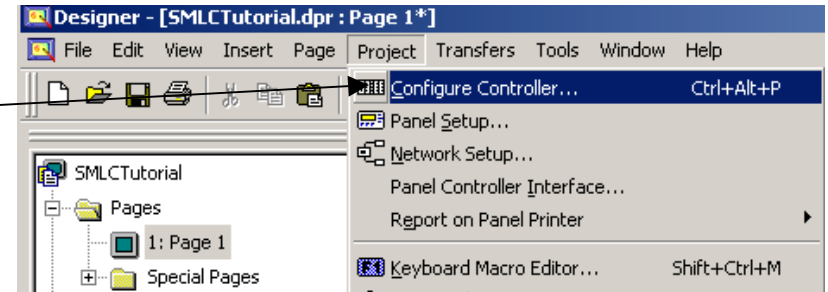
SMLC Configuration - Floating point variables

- In our HMI we want to have some floating point devices for displaying or entering floating point values.
- Right-click on the HMI/Modbus Interface and append 20 REAL variables.
- Note that REAL register numbers increase by two because each 32-bit value is two 16-bit Modbus registers.
- Double click on the AT in front of %QW122 and enter the variable name to be used in the SMLC program. Note that in our naming convention we use r to indicate that its a REAL for a variable name of hmi_rOutputValue1.
- Add another variable at %QW124 called hmi_rInputValue1.



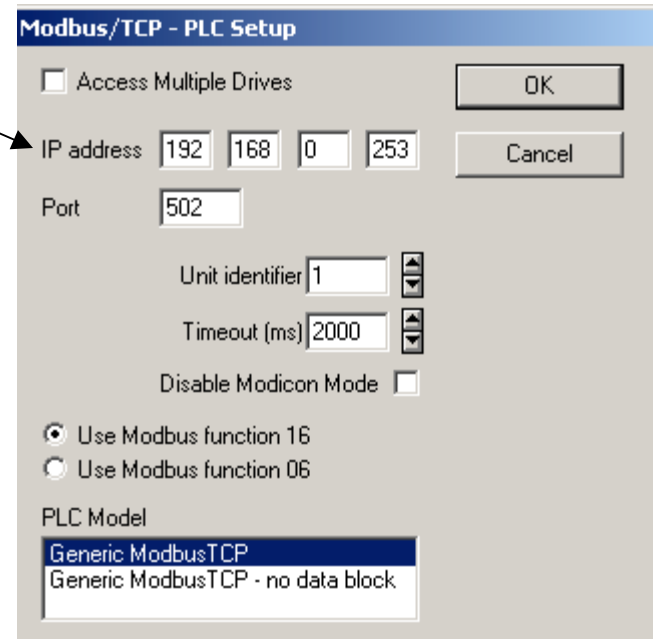
EXOR Configuration for operation with the SMLC

- Start the EXOR Designer software
- Create a new project
- Select Project | Configure Controller
- Select the Modbus/TCP driver
- Click on the Controller Setup button
- Click on ... to configure the Modbus/TCP driver.



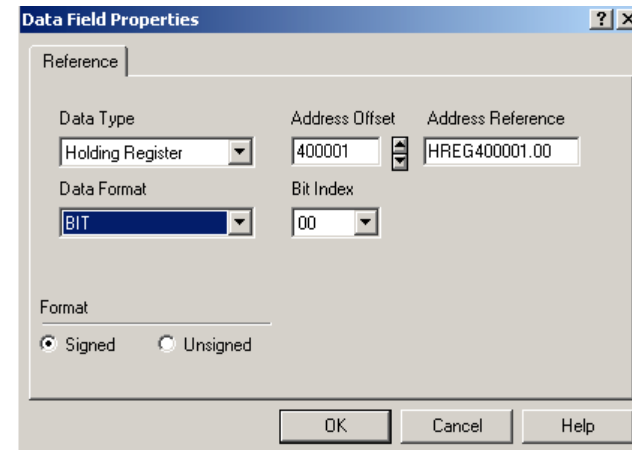
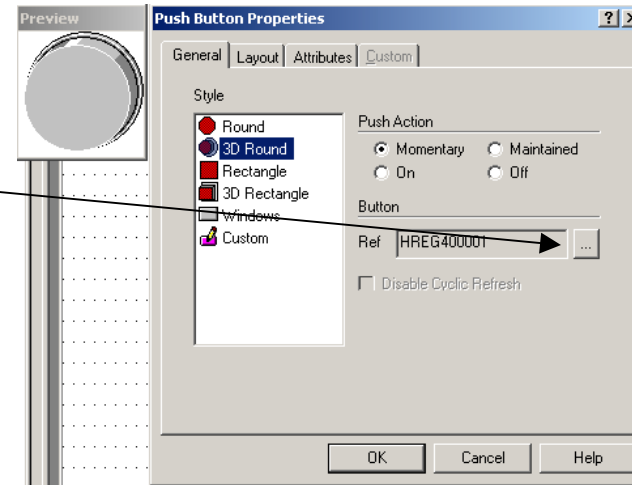
EXOR Configuration for operation with the SMLC

- Enter the IP Address of the SMLC
 - Leave the Port at 502
 - Leave the Unit identifier at 1
 - Set the desired timeout (the default of 2000 msec is fine)
 - Leave the Disable Modicon Mode checkbox unchecked
 - Make sure **Use Modbus function 16** is selected.
 - Select Generic ModbusTCP for the PLC Model.
 - Click on the OK button
- NOTE: This tutorial assumes that you have already configured the Ethernet port of the EXOR display. If not, refer to the EXOR documentation.



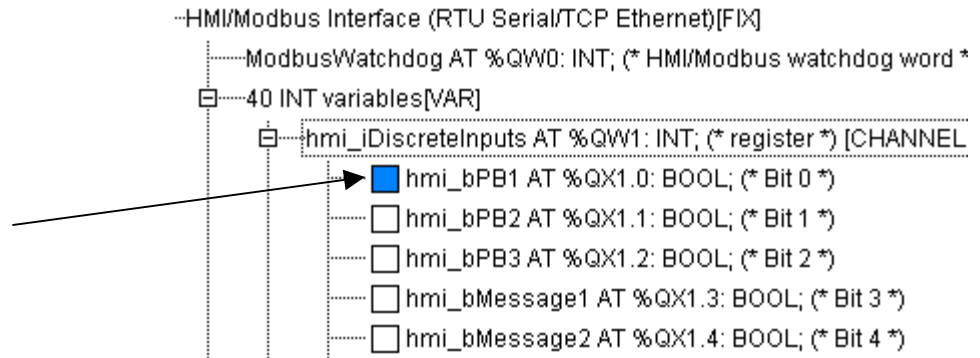
Single bit devices - Push button

- In Designer insert a Push Button onto the page.
- Click on the ... next to the Ref to bring up the Data Field Properties page.
- Select Data Type Holding Register
- Set the Address Offset to 400001 (remember this corresponds to %QW1 in the SMLC)
- Set the Data Format to BIT
- Set the Bit index to 0. This refers to bit 0 in register 1 or %QX1.0 in the SMLC PLC Configuration.



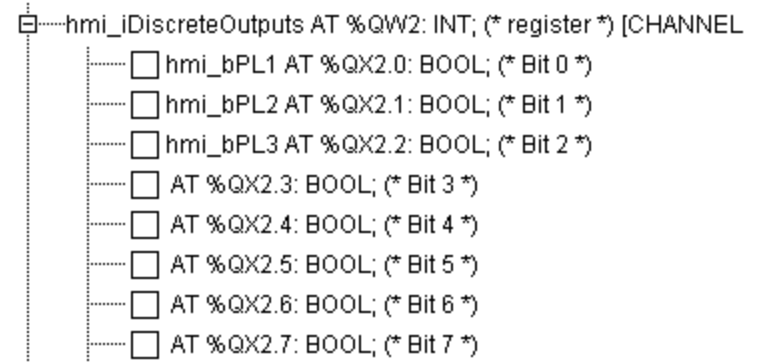
Single bit devices - Push button

- Download the CoDeSys project to the SMLC and run the program.
- Download the HMI project to the HMI.
- Press the Push Button on the HMI
- In the CoDeSys PLC Configuration you should see the input bit light up



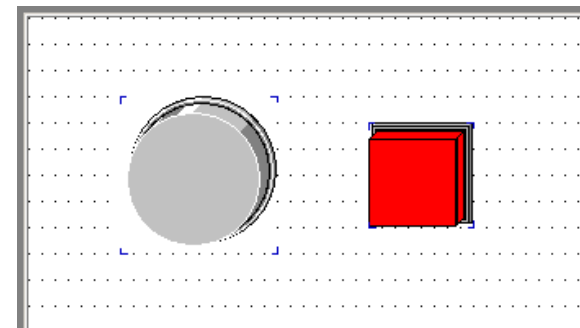
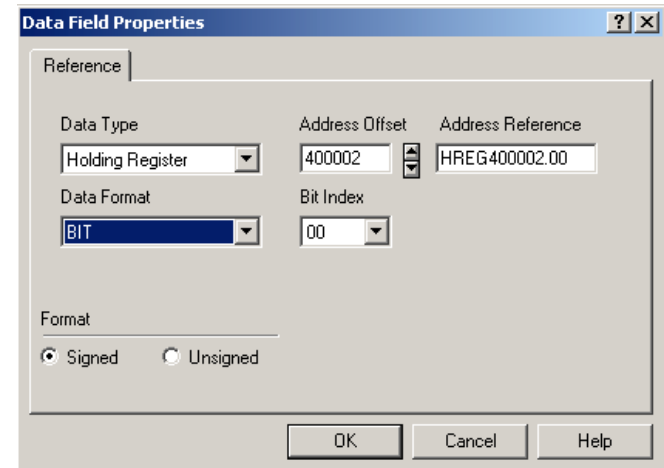
Single bit devices - Pilot Light

- Now lets create a Pilot light output.
- Recall that in the SMLC PLC Configuration we named bit 0 of Register 2 (%QX2.0) hmi_bPL1 (for Pilot Light 1)
- Add a run of logic to PLC_PRG that reads hmi_bPB1 and turns on hmi_bPL1
- Download this change to the SMLC



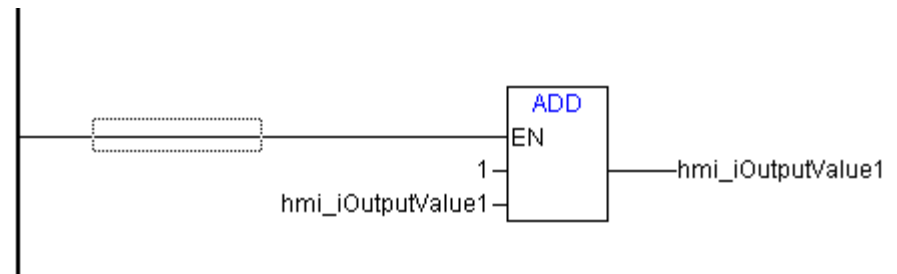
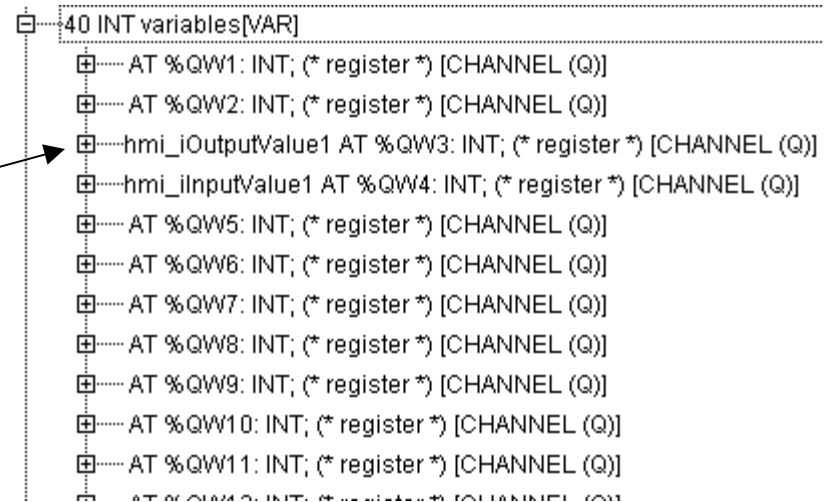
Single bit devices - Pilot Light

- In Designer insert an Indicator Light (Pilot light)
- On the Data Field Properties page set the Data Type to Holding Register
- Set the Address Offset to 400002 (which corresponds to %QW2 in the SMLC PLC Configuration)
- Set the Data Format to BIT
- Set the Bit Index to 00 (which corresponds to %QX2.0 in the SMLC PLC Configuration).
- Download this new page to the HMI
- Press the button on the screen. The Pilot light should turn Green and the PLC logic should indicate that the hmi_bPL1 output is turned on.
- **EXCERSISE** add the display elements and logic for two more pushbuttons and two pilot lights using the additional PB inputs and PL outputs that we defined earlier.



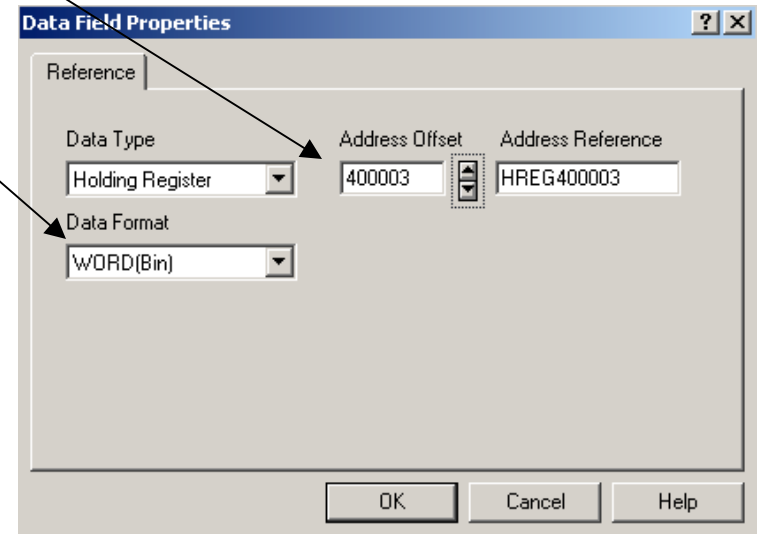
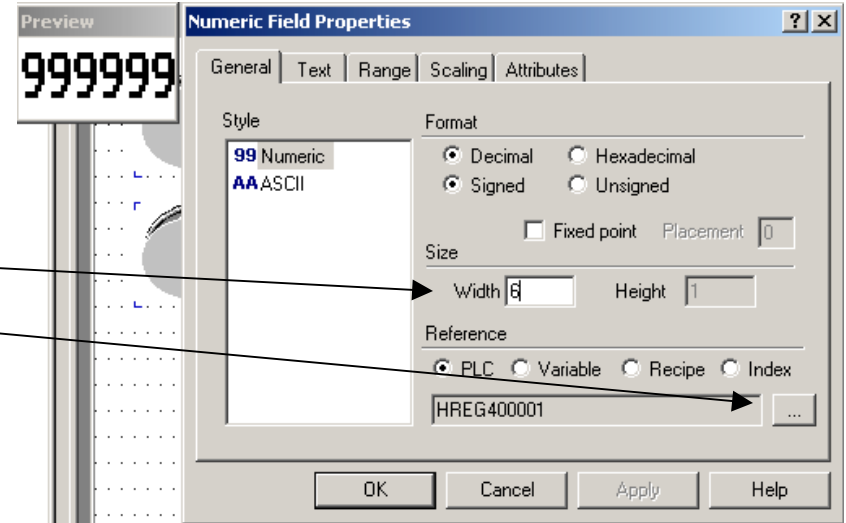
Integer data display

- In the next example we want to display an integer value from the SMLC program on the HMI.
- Recall that in the CoDeSys PLC Configuration we created an integer variable at %QW3 called hmi_iOutputValue1. This will be a value that we update in our PLC program and display on the HMI.
- In PLC_PRG add a rung of logic that adds 1 to hmi_iOutputValue1 and stores the result in hmi_iOutputValue1. This will cause hmi_iOutputValue1 to increment until it reaches 32767 at which point it will wrap around to -32768 and start incrementing again. We want to see this value incrementing on the HMI.
- Download this change to the SMLC.



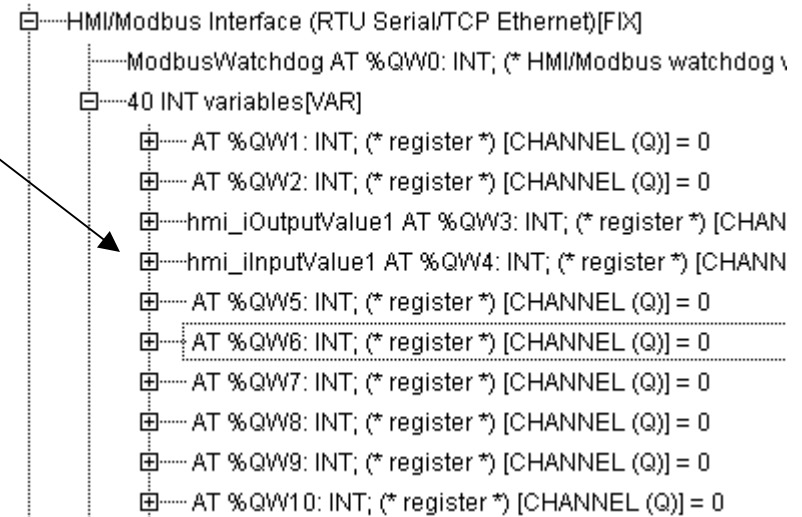
Integer data display

- In Designer insert a Numeric Field onto the page.
- Set the Width to 6
- Press the ... to edit the Data Field Properties.
- On the Data Field Properties page set the Address Offset to 400003 (corresponding to %QW3 in the SMLC's PLC Configuration)
- Set the Data Format to WORD(Bin)
- Download the screen to the HMI and verify that the value is updating.



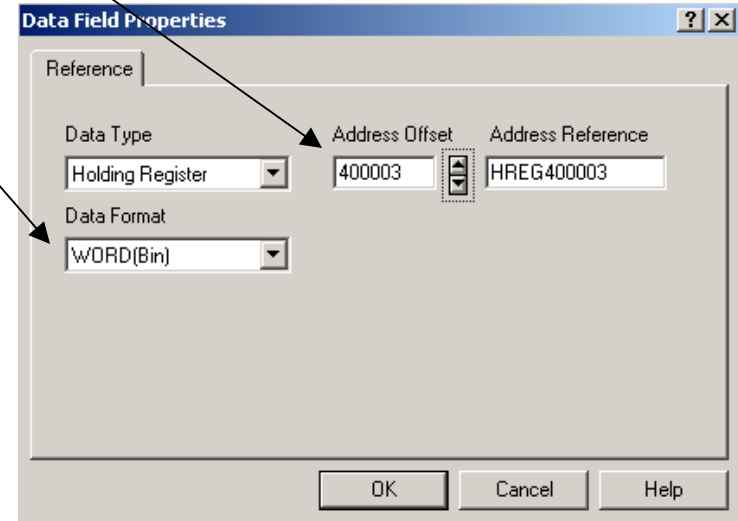
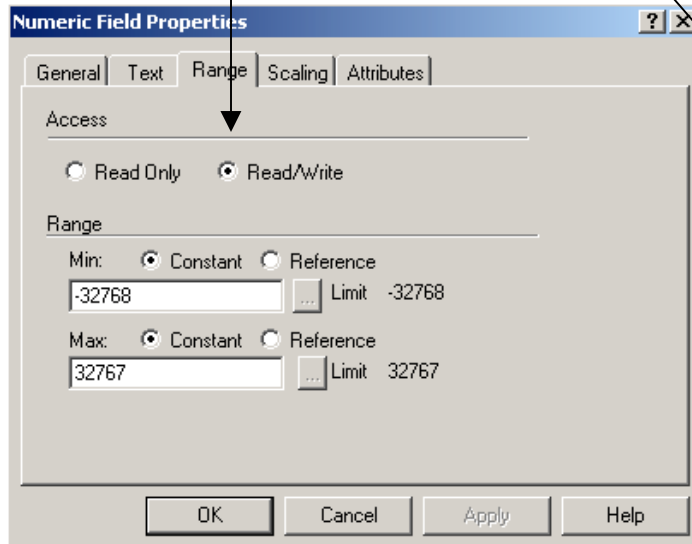
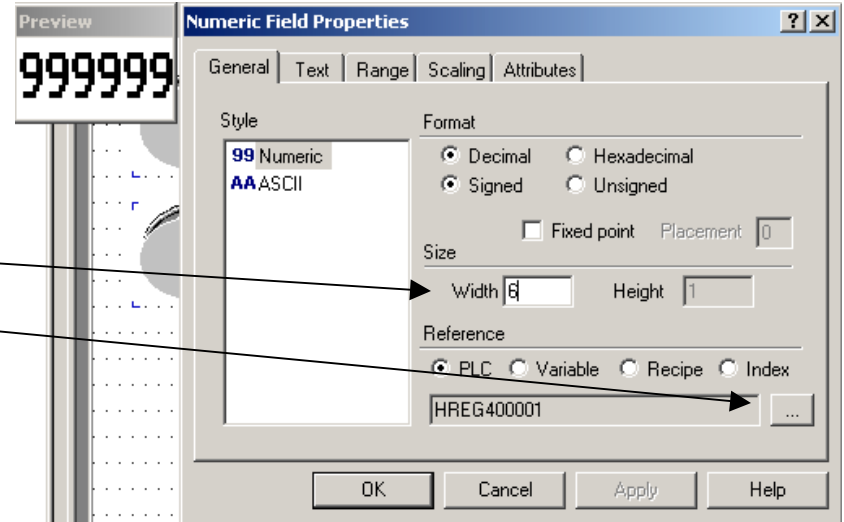
Integer data input

- In the next example we want to enter an integer data value on the HMI and have it appear in the SMLC program.
- Recall that in the CoDeSys PLC Configuration we created an integer variable at %QW4 called hmi_iInputValue1.
- The value entered on the HMI will show up here.



Integer data input

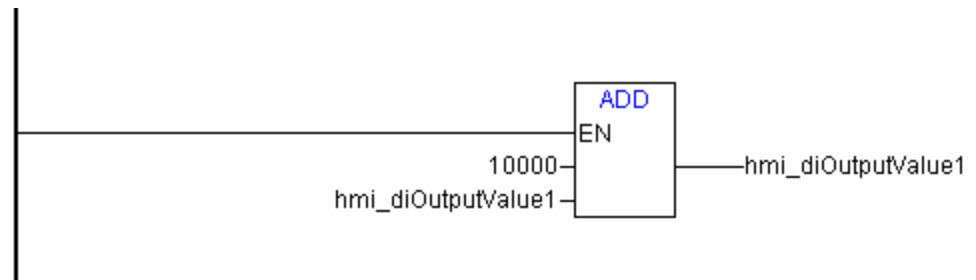
- In Designer insert a Numeric Field onto the page and set the Width to 6
- Press the ... to edit the Data Field Properties.
- On the Data Field Properties page set the Address Offset to 400003 (corresponding to %QW3 in the SMLC's PLC Configuration).
- Set the Data Format to WORD(Bin).
- On the Range tab set the Access to Read/Write



Double Integer data display

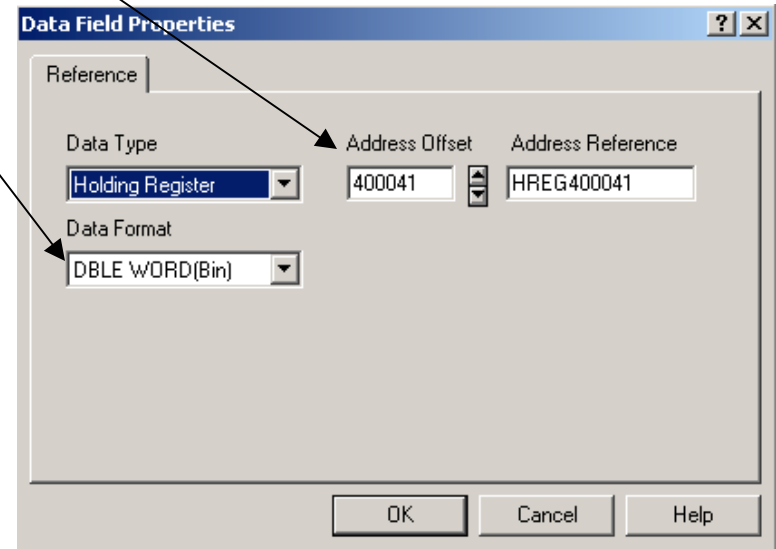
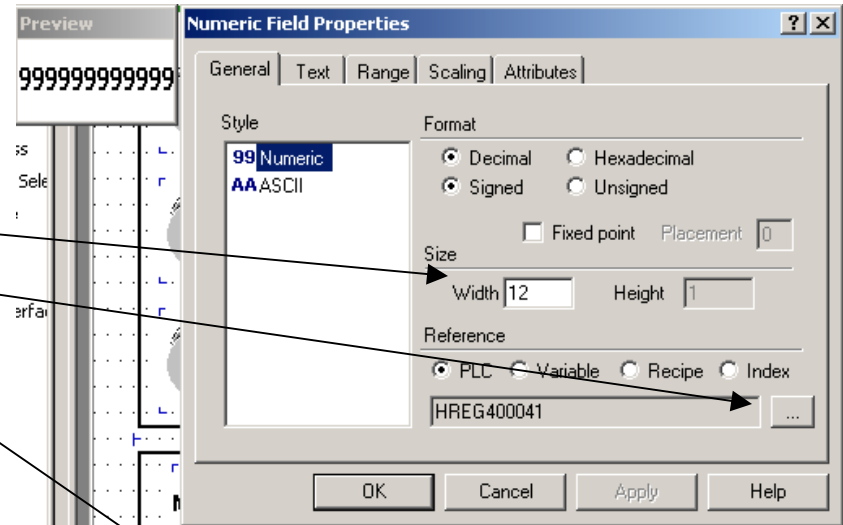
- In the next example we want to display an integer value from the SMLC program on the HMI.
- Recall that in the CoDeSys PLC Configuration we created an integer variable at %QW41 called hmi_diOutputValue1. This will be a value that we update in our PLC program and display on the HMI.
- In PLC_PRG add a run that adds 10000 to hmi_diOutputValue1 and stores the result in hmi_diOutputValue1. This will cause hmi_diOutputValue1 to increment until it reaches 2147483647 at which point it will wrap around to -2147483648 and start incrementing again.
- Download this to the SMLC

```
20 DINT variables[VAR]
-----hmi_diOutputValue1 AT %QW41: DINT; (* registers *) [CHANNEL (Q)]
-----hmi_diInputValue1 AT %QW43: DINT; (* registers *) [CHANNEL (Q)]
-----AT %QW45: DINT; (* registers *) [CHANNEL (Q)]
-----AT %QW47: DINT; (* registers *) [CHANNEL (Q)]
-----AT %QW49: DINT; (* registers *) [CHANNEL (Q)]
-----AT %QW51: DINT; (* registers *) [CHANNEL (Q)]
-----AT %QW53: DINT; (* registers *) [CHANNEL (Q)]
-----AT %QW55: DINT; (* registers *) [CHANNEL (Q)]
-----AT %QW57: DINT; (* registers *) [CHANNEL (Q)]
-----AT %QW59: DINT; (* registers *) [CHANNEL (Q)]
```



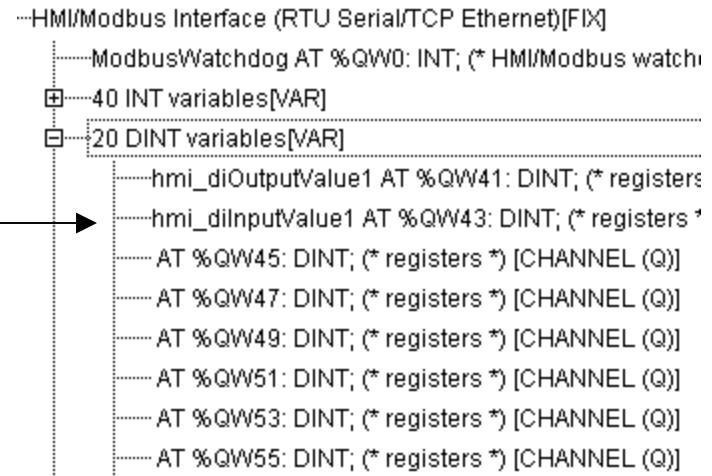
Double Integer data display

- In Designer insert a Numeric Field onto the page
- Set the Width to 12
- Press the ... to edit the Data Field Properties
- On the Data Field Properties page set the Address Offset to 400041 (corresponding to %QW41 in the SMLC's PLC Configuration)
- Set the Data Format to DBLE WORD(Bin)
- Download this program to the HMI and verify that the value updates.



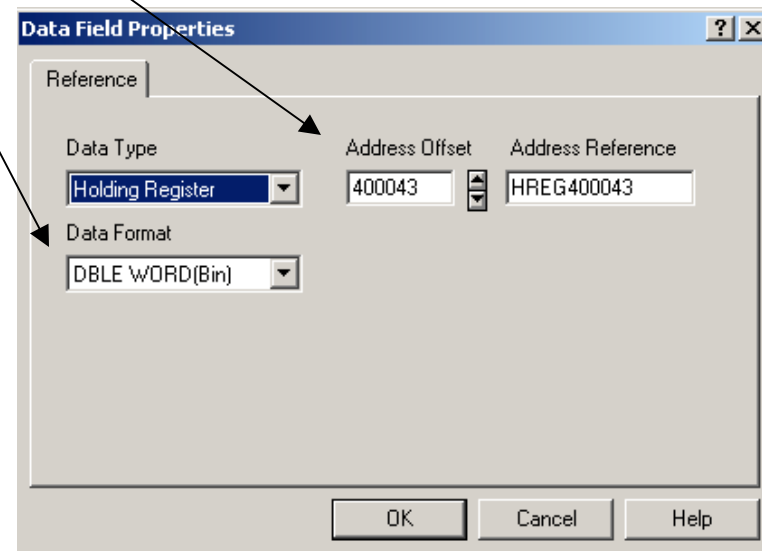
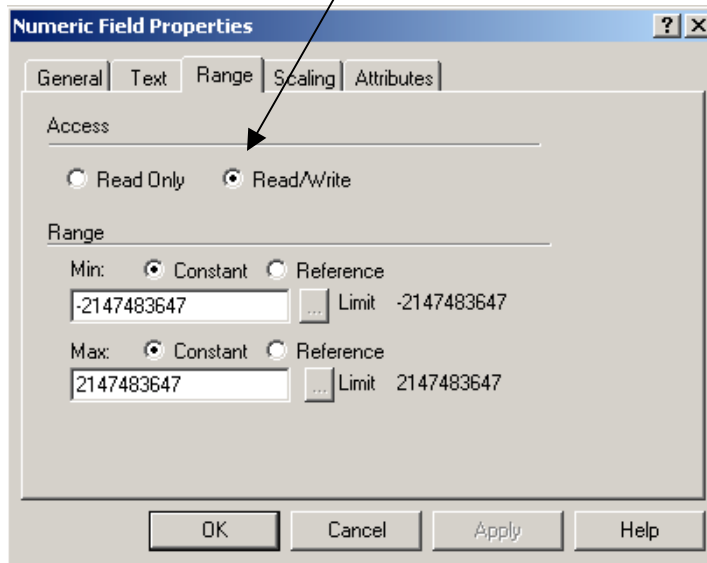
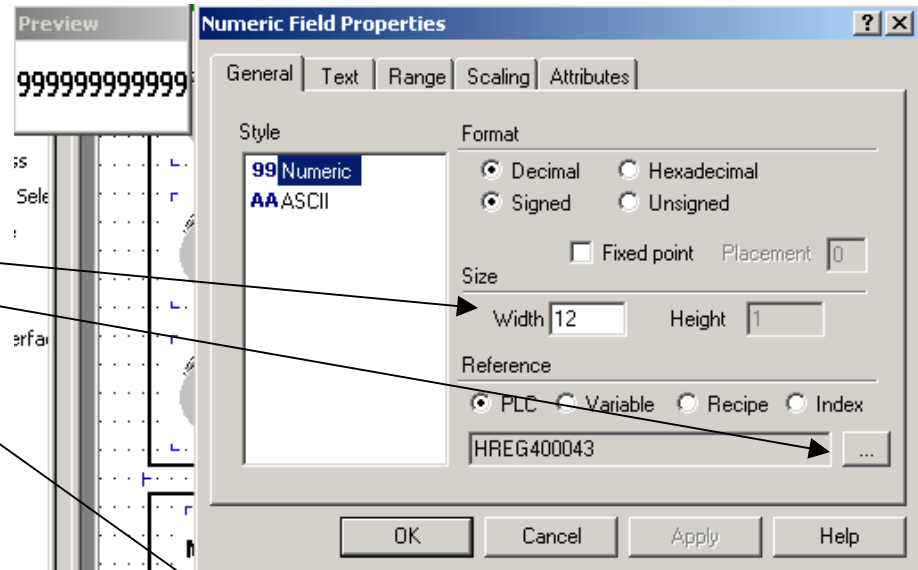
Double Integer data input

- In the next example we want to enter an integer data value on the HMI and have it appear in the SMLC program.
- Recall that in the CoDeSys PLC Configuration we created a double integer variable at %QW43 called `hmi_diInputValue1`.
- The value entered on the HMI will show up here.



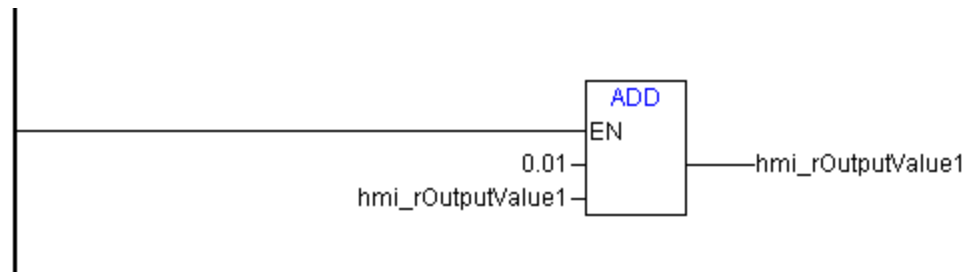
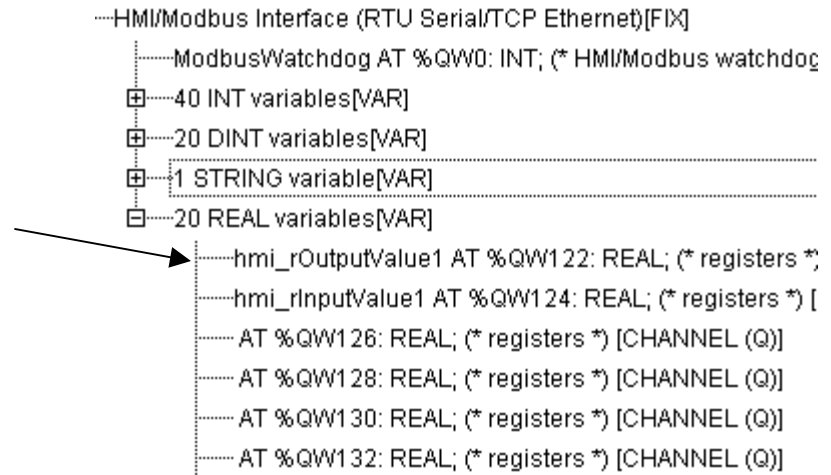
Double Integer data input

- In Designer insert a Numeric Field onto the page
- Set the Width to 12 and then press the ... to edit the Data Field Properties
- On the Data Field Properties page set the Address Offset to 400043 (corresponding to %QW43 in the SMLC's PLC Configuration)
- Set the Data Format to DBLE WORD(Bin)
- On the Range tab enable Read/Write



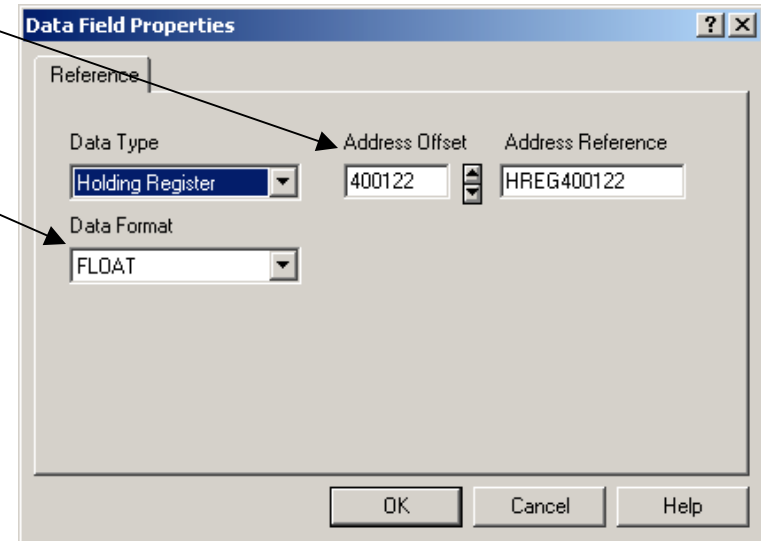
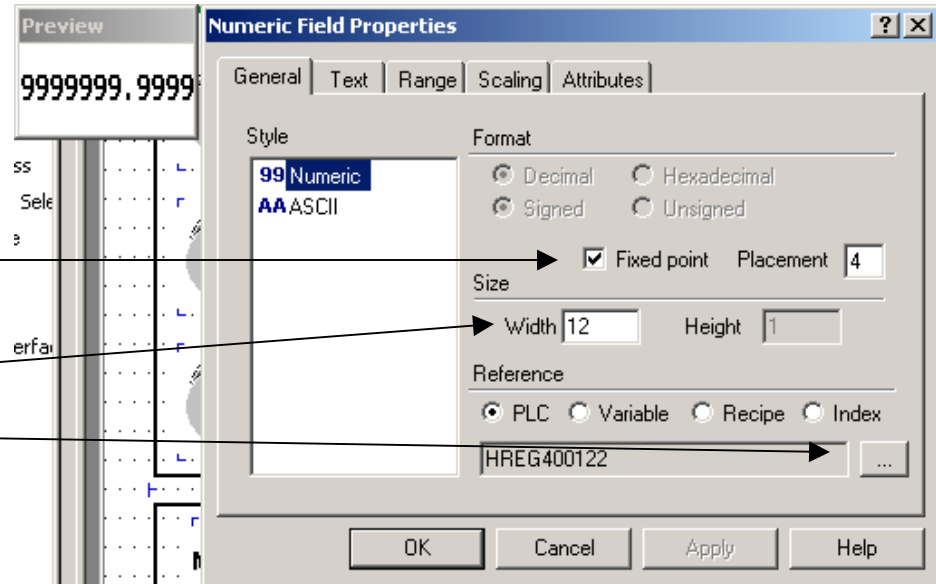
Floating point data display

- In the next example we want to display an integer value from the SMLC program on the HMI.
- Recall that in the PLC Configuration we created a real variable hmi_rOutputValue1 at %QW122. This will be a value that we update in our PLC program and display on the HMI.
- In PLC_PRG add a run that adds .01 to hmi_rOutputValue1 and stores the result in hmi_rOutputValue1. This will cause hmi_rOutputValue1 to increment until it reaches the maximum floating point value at which point it will wrap around to the minimum floating point value.



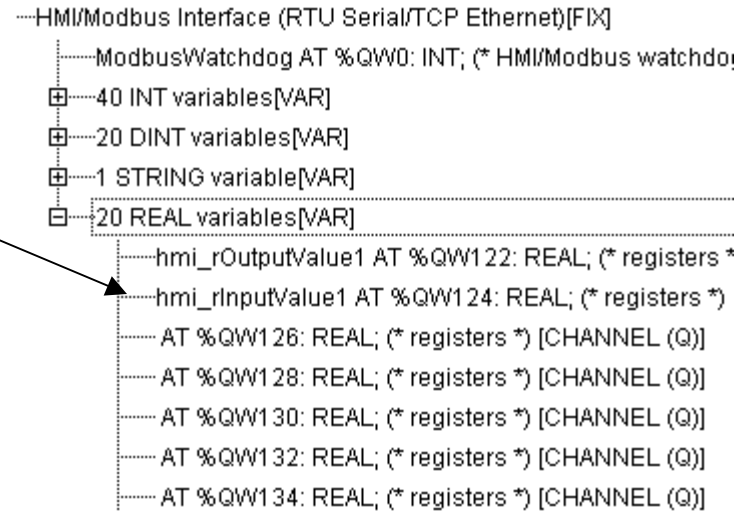
Floating point data display

- In Designer insert a Numeric Field onto the page
- Set the Format to Fixed point and select the desired number of decimal places, in this example 4.
- Set the Width to 12
- Press the ... to edit the Data Field Properties
- On the Data Field Properties page set the Address Offset to 400122 (corresponding to %QW122 in the SMLC's PLC Configuration)
- Set the Data Format to FLOAT



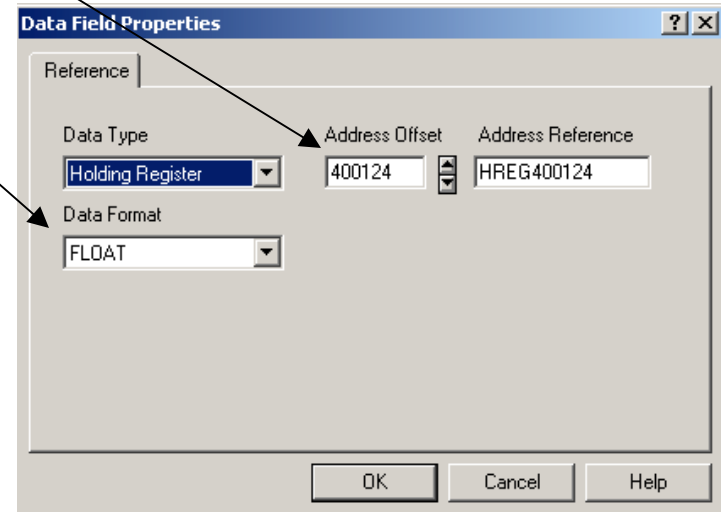
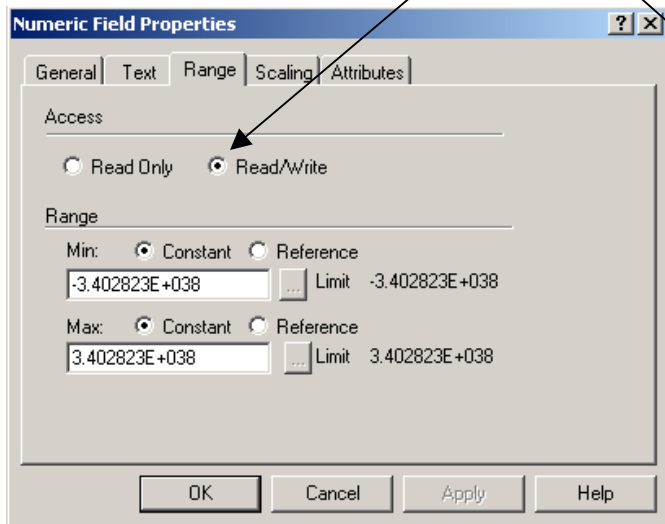
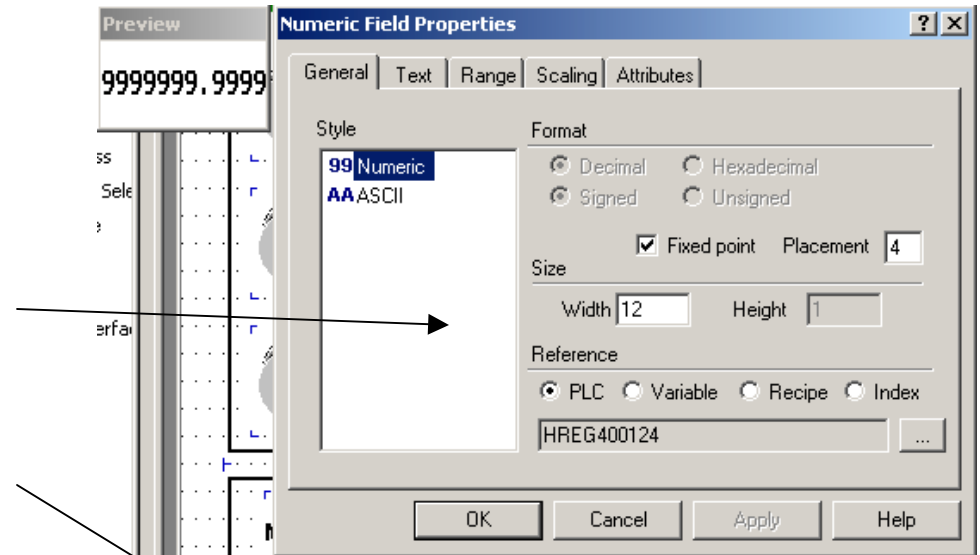
Floating point data input

- In the next example we want to enter an integer data value on the HMI and have it appear in the SMLC program.
- Recall that in the CoDeSys PLC Configuration we created a real variable at %QW124 called hmi_rInputValue1. The value entered on the HMI will show up here.



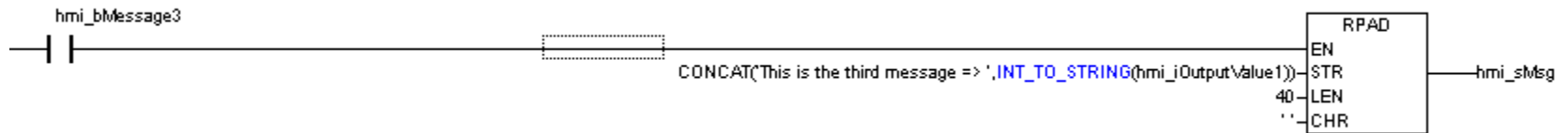
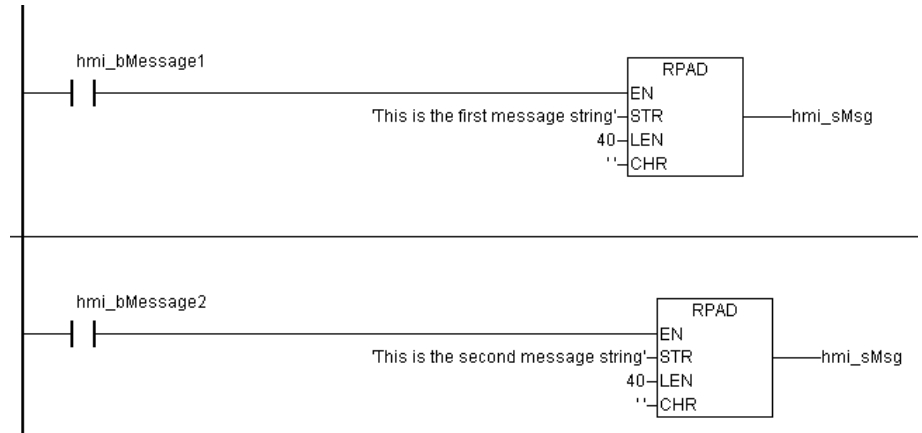
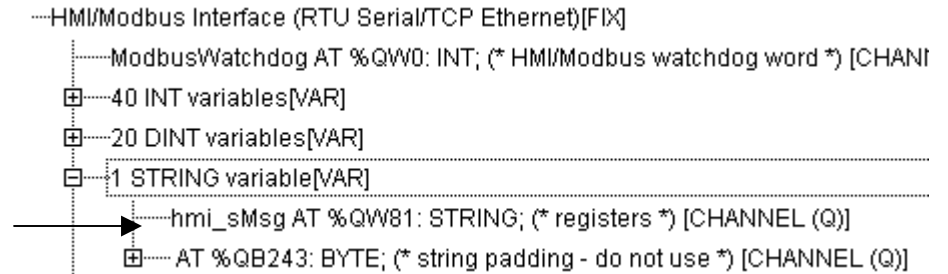
Floating point data input

- In Designer insert a Numeric Field onto the page
- Set the Width to 12, enable Fixed point, set the Placement to 4 and then press the ... to edit the Data Field Properties
- On the Data Field Properties page set the Address Offset to 400124 (corresponding to %QW124 in the SMLC's PLC Configuration)
- Set the Data Format to FLOAT
- On the Range tab enable Read/Write



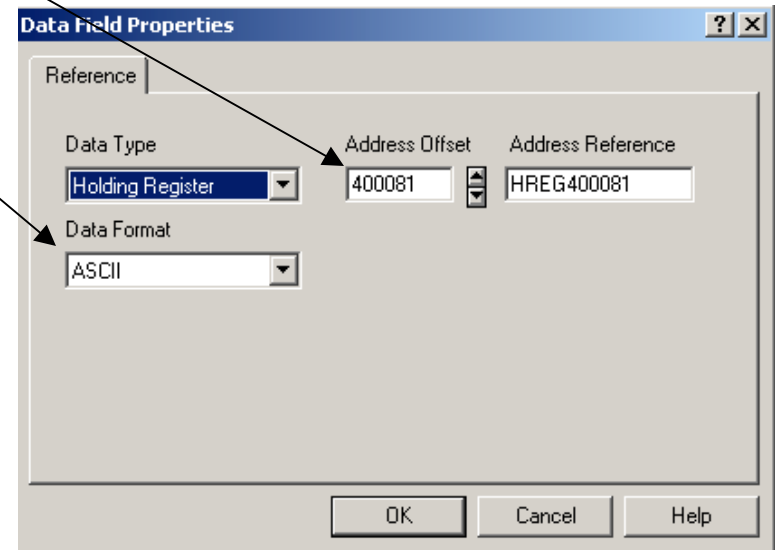
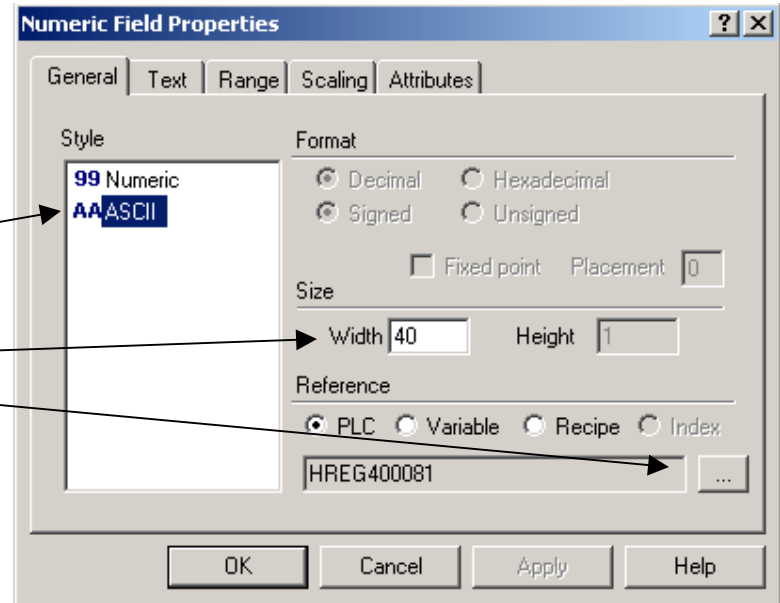
String data display

- In the next example we want to display a string value from the SMLC program on the HMI.
- Recall that in the CoDeSys PLC Configuration we created a string variable at %QW81 called hmi_sMsg. This will be a value that we update in our PLC program and display on the HMI.
- In PLC_PRG add logic that fills the hmi_sMsg variable with three different strings depending on which input condition is true. We use the RPAD function from the OrmLibMisc library rather than a simple MOVE block so we can pad the end of the string with spaces out to the maximum string length supported by the EXOR of 40 characters.
- The 3rd string we make dynamic by concatenating an ever changing value.



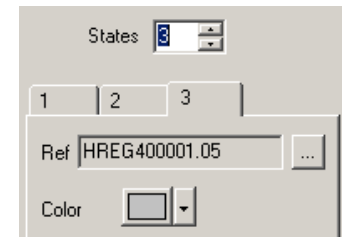
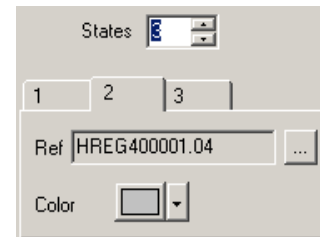
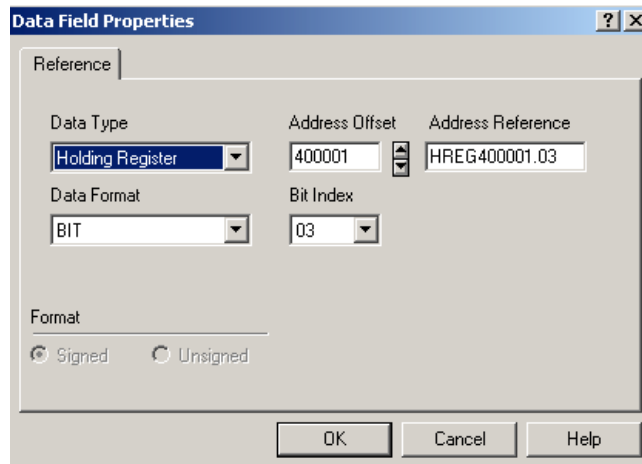
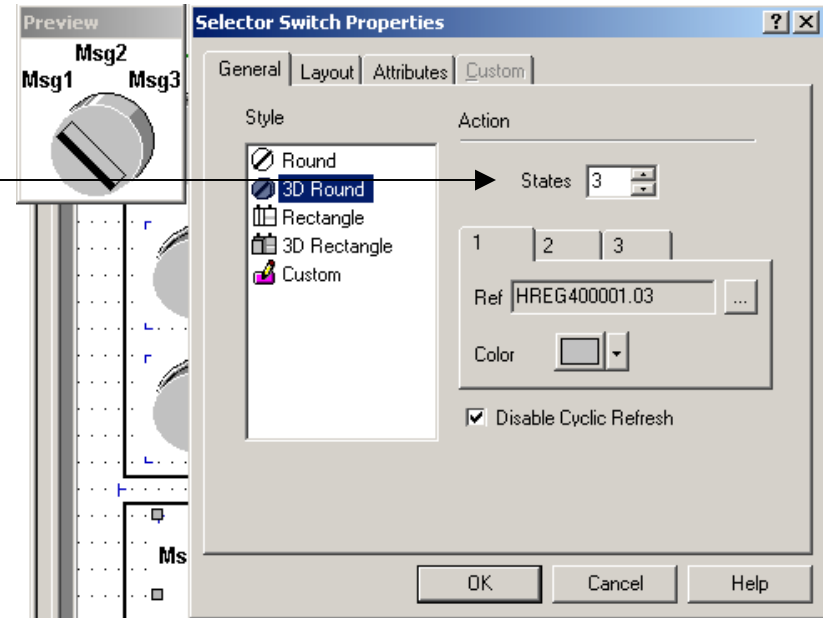
String data display

- In Designer insert a Numeric Field onto the page
- Select Style ASCII
- Set the Width to 40 and then press the ... to edit the Data Field Properties
- On the Data Field Properties page set the Address Offset to 400081 (corresponding to %QW81 in the SMLC's PLC Configuration)
- Set the Data Format to ASCII

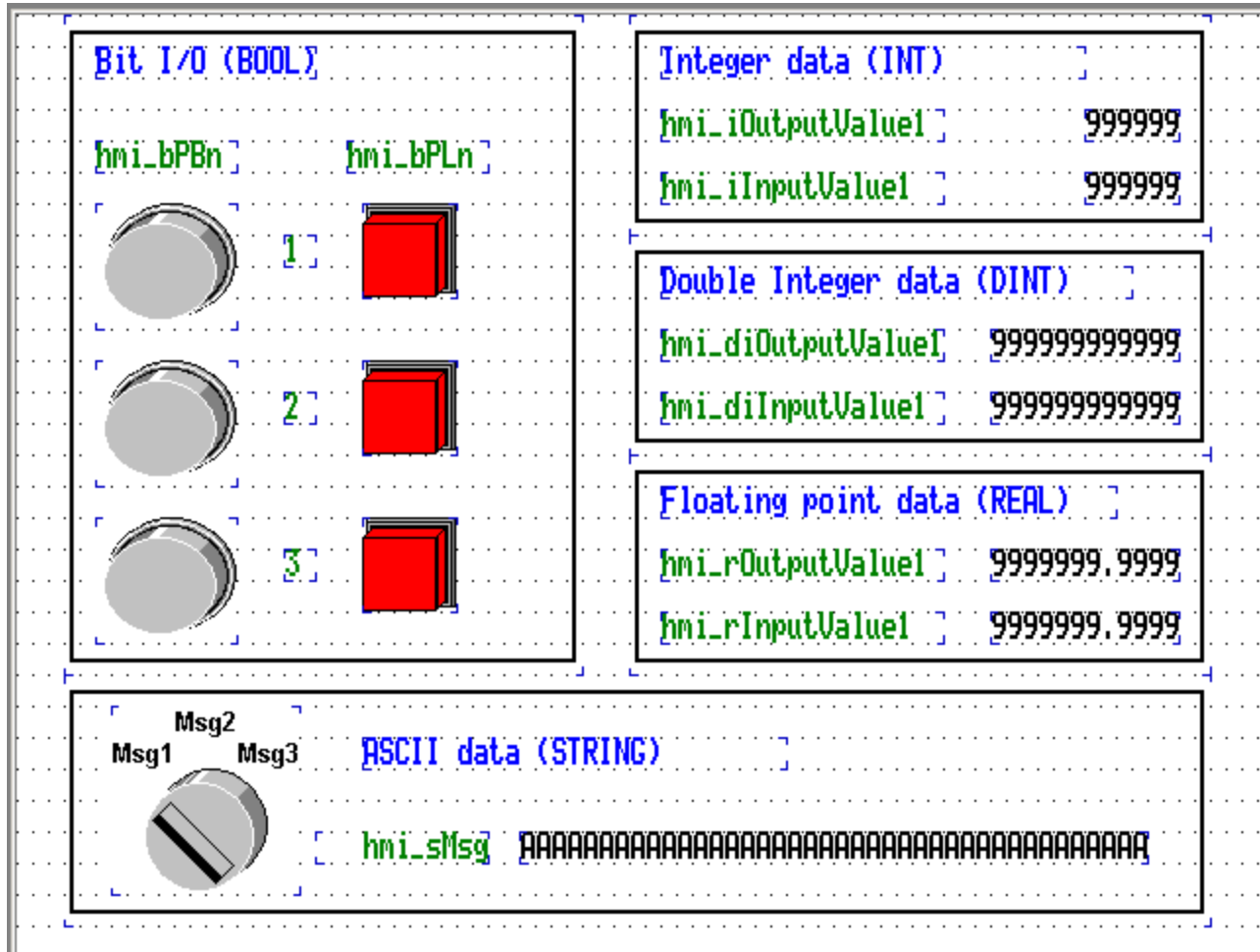


String data display - adding a selector switch

- Now lets add a selector switch to choose between the three string messages
- In Designer insert a selector switch onto the page.
- Set the number of states to 3
- For each state go to the Data Field Properties and set the Address offset to 400001 (%QW1), select BIT for the Data Format and set the Bit Offset at 3, 4 and 5 respectively for the three states. This corresponds to %QX1.3, %QX1.4 and %QX1.5 in the SMLC PLC Configuration



Our final screen layout

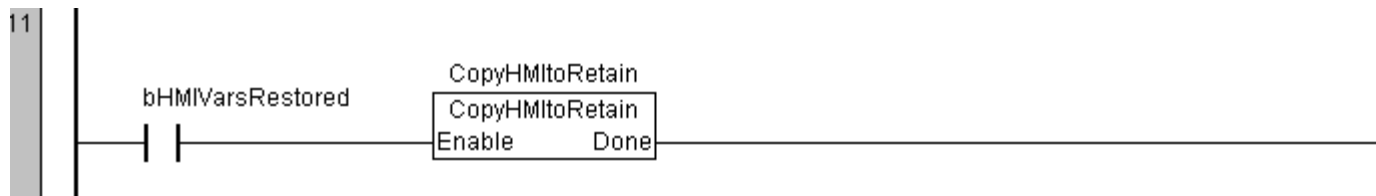


Dealing with Retain/Persistent HMI variables

- The SMLC initializes all of its I/O in the PLC Configuration at powerup including the HMI registers.
- If we want to have HMI variables be retained and/or persistent we will need to copy the HMI variable to retain/persistent variables every scan of our program and we will also need to copy the retain/persistent variables to our HMI variables once after powerup.
- To make this simpler we will create two user-defined function blocks: CopyHMItoRetain and CopyRetainToHMI.
- For this example the variables we will be concerned with are hmi_iInputValue1, hmi_diInputValue1 and hmi_rInputValue1 but this example can easily be extended to include other HMI variables.
- In the first rung of our program we will copy the retain/persistent variables to the HMI variables on the rising edge of the Execute input (which will occur on the first scan of our program after powerup).

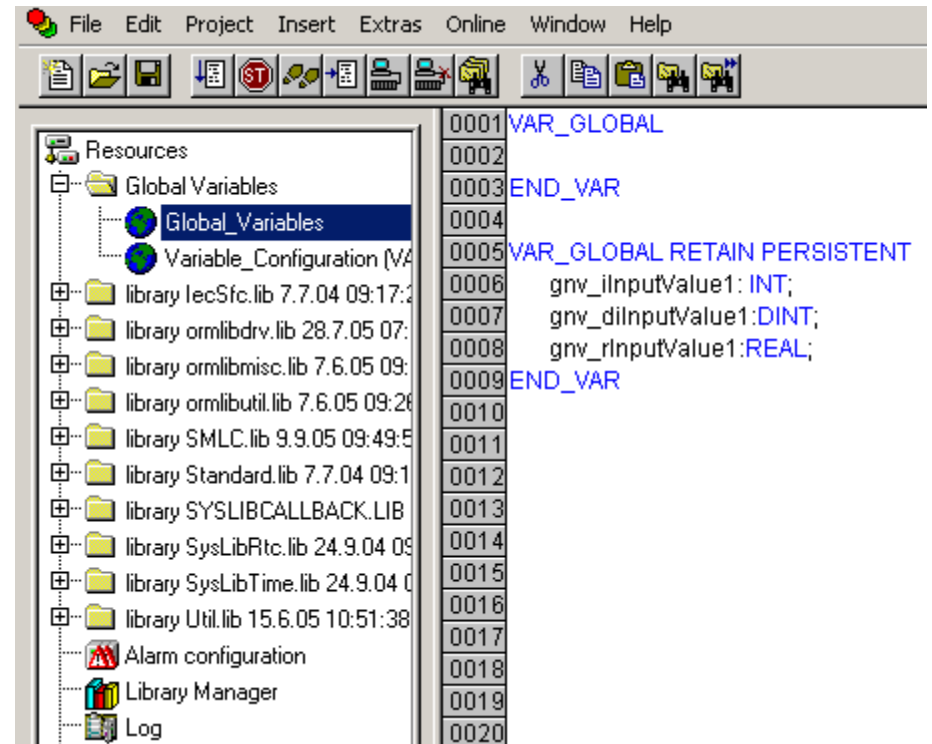


- In the last rung of our program we will copy the HMI variables to the retain/persistent variables. Note that we do this every scan after the CopyRetainToHMI as completed.



Declaring the retain/persistent variables

- On the Global Variables section of the Resources tab declare our three variables in a section as shown.
- By convention we prefix the variable names with gnv_ which stands for “global non-volatile”.



CopyRetainToHMI Function Block source

- Lets examine the CopyRetainToHMI function block
- On the rising edge of the Execute input (which will happen on the very first scan of our program) start a 100 msec timer.
- The timer is required because the SMLC initializes all of its inputs at powerup and this is done by another task and may not be completed by the time our first scan runs. By delaying 100msec we can be certain that this task has completed.
- After the timeout has completed we copy the contents of the HMI variables to the retain variables and then turn on the Done output. You should use this Done output to control when you start copying the HMI variables into the retain variables.

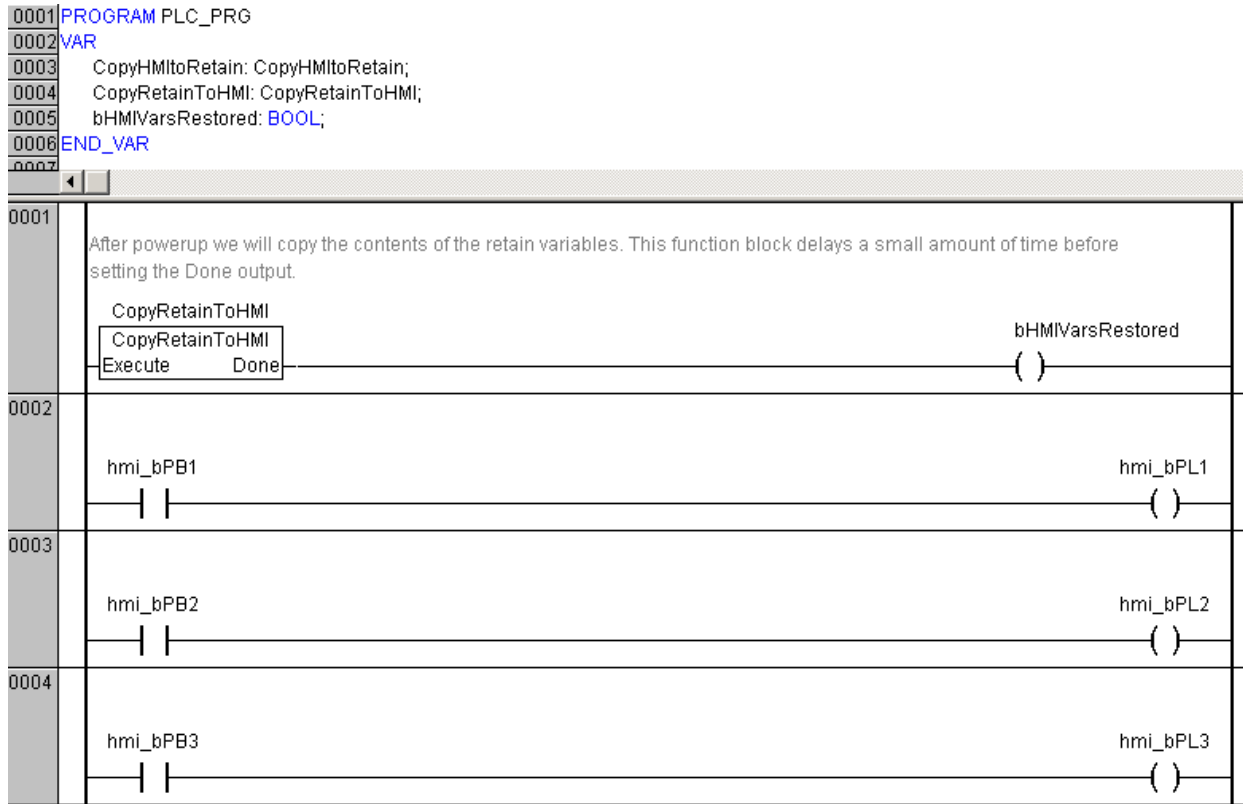
```
0001 FUNCTION_BLOCK CopyRetainToHMI
0002 VAR_INPUT
0003     Execute:BOOL;
0004 END_VAR
0005 VAR_OUTPUT
0006     Done:BOOL;
0007 END_VAR
0008 VAR
0009     bOldExecute:BOOL:=FALSE;
0010     tonWait : TON;
0011     bStarted:BOOL := FALSE;
0012 END_VAR
0013
0001 IF Execute AND NOT bOldExecute THEN
0002     tonWait(IN:=TRUE , PT:=T#100ms);
0003     bStarted := TRUE;
0004 END_IF
0005 tonWait();
0006 IF tonWait.Q AND bStarted THEN
0007     hmi_iInputValue1 := gnv_iInputValue1;
0008     hmi_dInputValue1 := gnv_dInputValue1;
0009     hmi_rInputValue1 := gnv_rInputValue1;
0010     bStarted := FALSE;
0011 END_IF
0012 Done:=Execute AND tonWait.Q;
0013 bOldExecute:=Execute;
0014
```

CopyHMIToRetain Function Block source

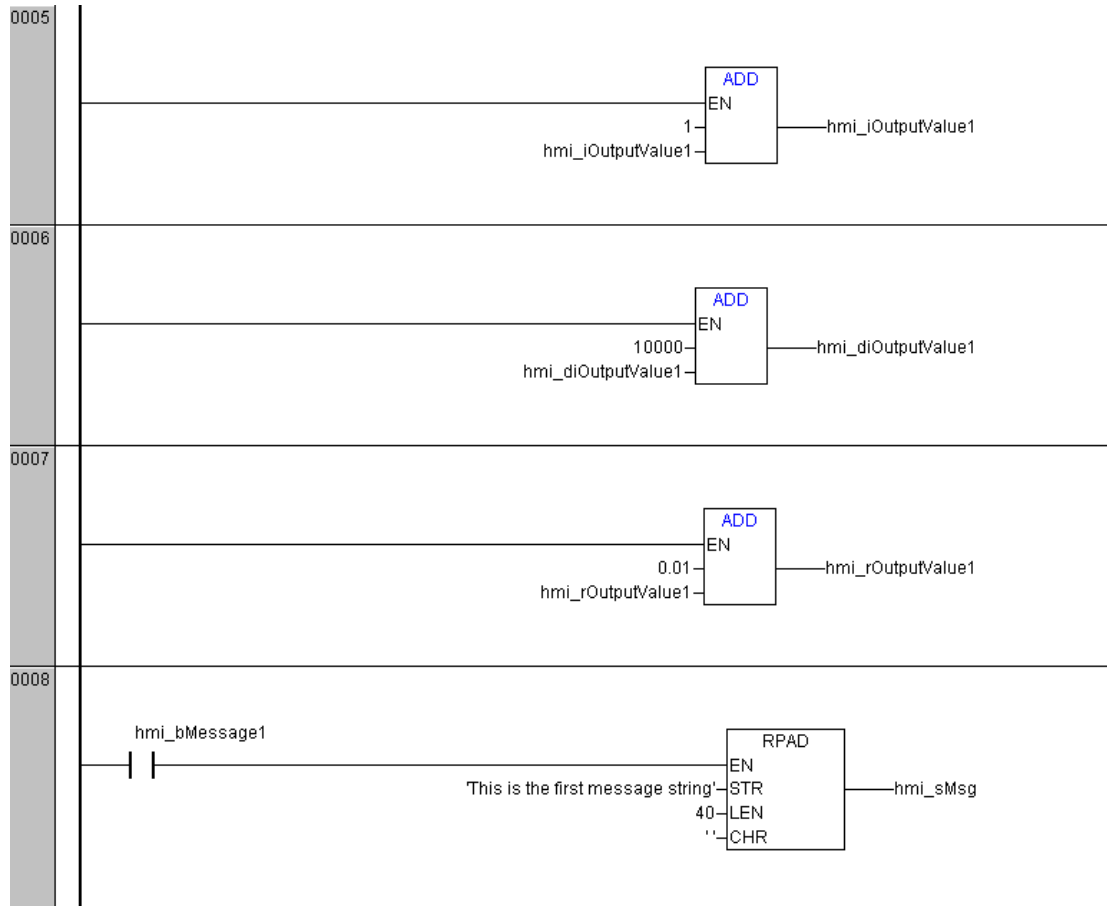
- Now lets examine the CopyHMIToRetain function block source.
- If the Enable input is True then we copy all HMI variables to their retain equivalents.
- You should do this every scan after the CopyRetainToHMI has completed.

```
0001 FUNCTION_BLOCK CopyHMIToRetain
0002 VAR_INPUT
0003     Enable:BOOL;
0004 END_VAR
0005 VAR_OUTPUT
0006     Done:BOOL;
0007 END_VAR
0008 VAR
0009 END_VAR
0010
0001 IF Enable THEN
0002     gnv_iInputValue1:= hmi_iInputValue1;
0003     gnv_dInputValue1:=hmi_dInputValue1;
0004     gnv_rInputValue1:=hmi_rInputValue1;
0005 END_IF
0006 Done:=Enable;
0007
0008
```

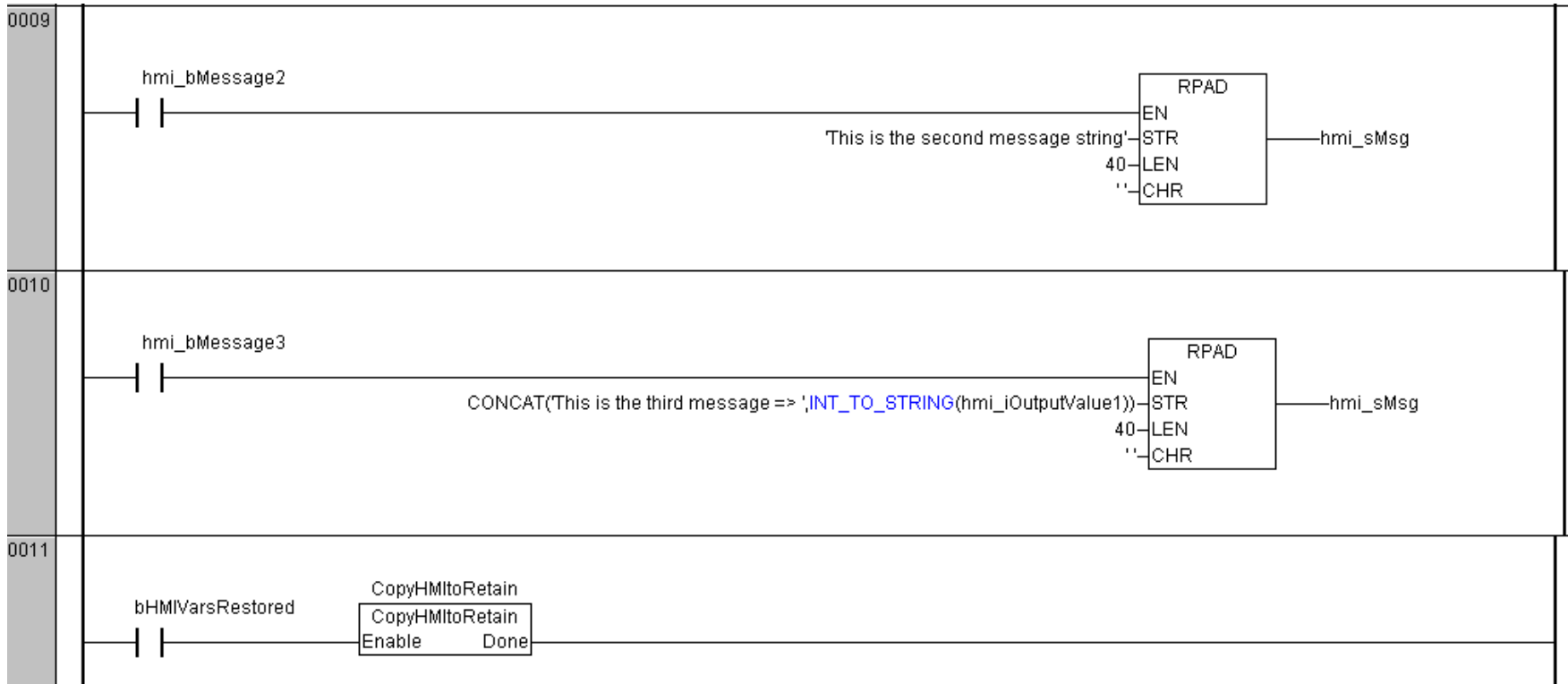
Final SMLC program source for PLC_PRG



Final SMLC program source for PLC_PRG



Final SMLC program source for PLC_PRG



Conclusion

- This concludes our SMLC/EXOR HMI tutorial



Ethernet (Modbus/TCP) connection

