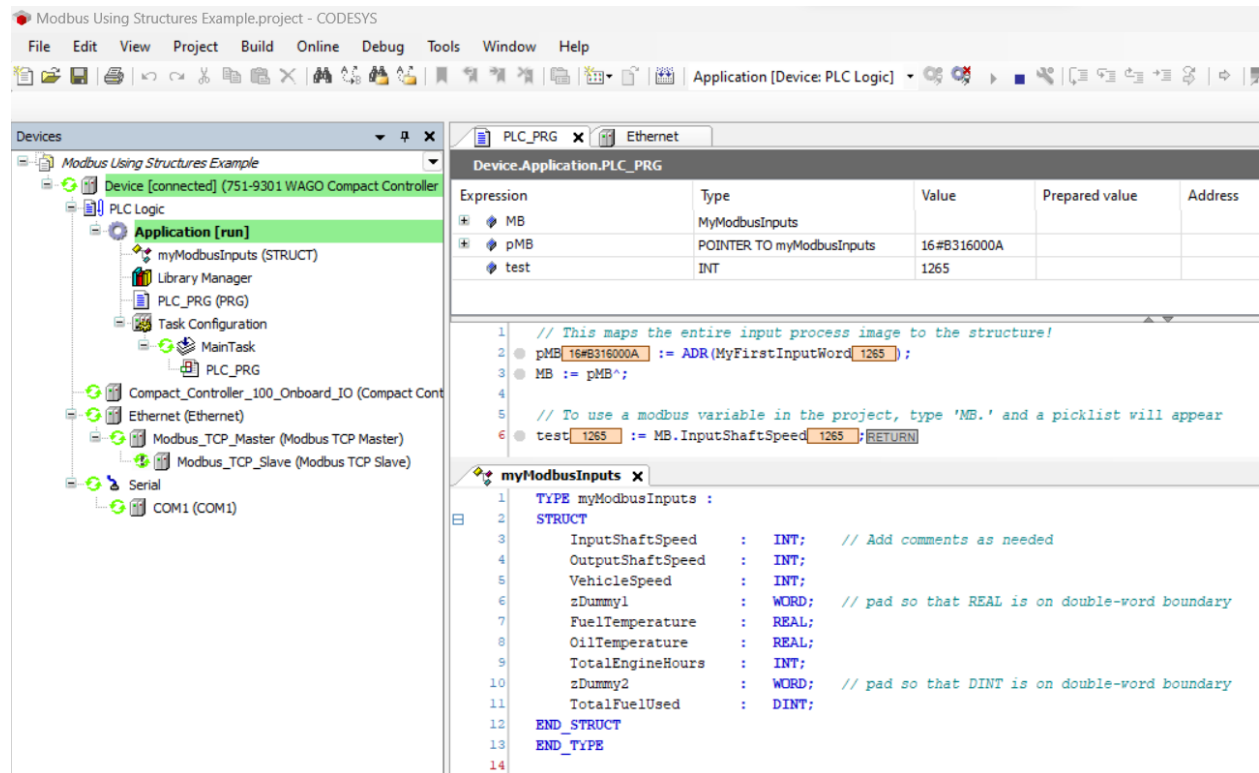


HOW TO MAP MODBUS VARIABLES USING STRUCTURES

Structures offer a convenient and time-saving means for creating, mapping, maintaining, and using Modbus variables in a CODESYS application.

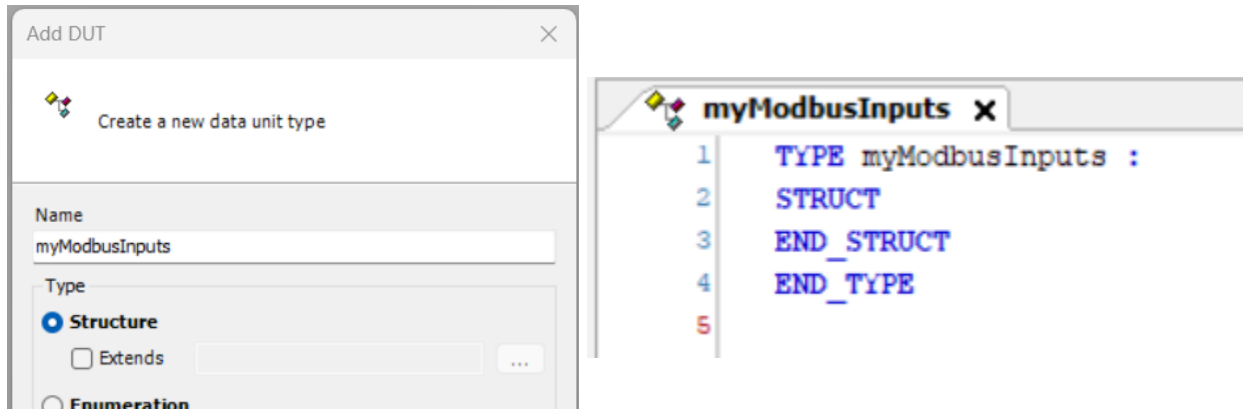
This example will show the advantages of mapping Modbus variables being read from a Modbus slave device using a structure.



The examples shown in this document were created using the following hardware & software:

Item Number	Description	SW or FW Level
-	CODESYS 3.5	V3.5 SP19 Patch 2
751-9301	WAGO Compact Controller 100	04.04.03(26)
750-862	WAGO Controller ModbusTCP (used as MB slave)	01.03.05(07)
		-

Start by creating a structure in CODESYS that will represent all Modbus data being read in a single transaction. Here a structure is created called *myModbusInputs*.



Create a list of the application's Modbus variables. These could be typed within CODESYS, but any editor can be used. Excel works well because its editing features help to quickly format each line.

	A	B	C	D	E	F
1						
2	0		InputShaftSpeed	:	INT;	// Add comments as needed
3	1		OutputShaftSpeed	:	INT;	
4	2		VehicleSpeed	:	INT;	
5	3		FuelTemperature	:	REAL;	
6	4		OilTemperature	:	REAL;	
7	5		TotalEngineHours	:	INT;	
8	6		TotalFuelUsed	:	DINT;	

Unfortunately, there is a problem. It is necessary that REAL and DINT (and some other) variables start at double-word boundaries. To accommodate this, add some dummy variables to occupy unused space. Depending on the type of data being transmitted, this may not be necessary at all, or it may be necessary in multiple places, as shown here:

	A	B	C	D	E	F	G	H
1								
2	0		InputShaftSpeed	:	INT;	// Add comments as needed		
3	1		OutputShaftSpeed	:	INT;			
4	2		VehicleSpeed	:	INT;			
5	3		zDummy1	:	WORD;	// pad so that REAL is on double-word boundary		
6	4		FuelTemperature	:	REAL;			
7	5		OilTemperature	:	REAL;			
8	6		TotalEngineHours	:	INT;			
9	7		zDummy2	:	WORD;	// pad so that DINT is on double-word boundary		
10	8		TotalFuelUsed	:	DINT;			
11								

These dummy variables can be named anything, but it's advantageous to name them starting with the letter 'z'. This will be explained later.

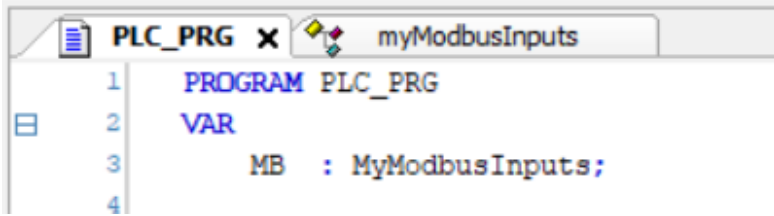
When the list is complete, copy it to the structure.

```

myModbusInputs x
1  TYPE myModbusInputs :
2  STRUCT
3      InputShaftSpeed      : INT;    // Add comments as needed
4      OutputShaftSpeed    : INT;
5      VehicleSpeed        : INT;
6      zDummy1             : WORD;    // pad so that REAL is on double-word boundary
7      FuelTemperature      : REAL;
8      OilTemperature       : REAL;
9      TotalEngineHours     : INT;
10     zDummy2             : WORD;    // pad so that DINT is on double-word boundary
11     TotalFuelUsed       : DINT;
12 END_STRUCT
13 END_TYPE
14

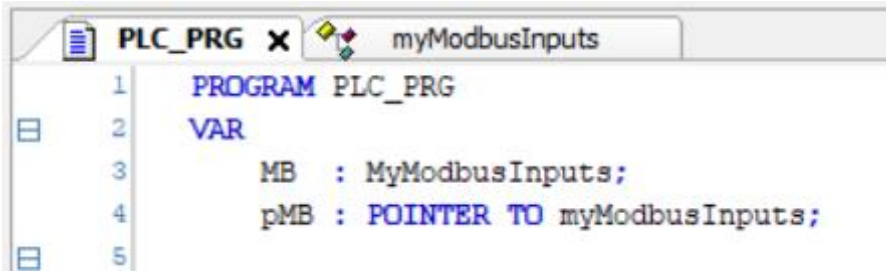
```

When the structure definition is complete, declare a variable of the structure type. Keep the name short, such as 'MB' or 'MBI' (for Modbus Inputs), because this name will be typed frequently.



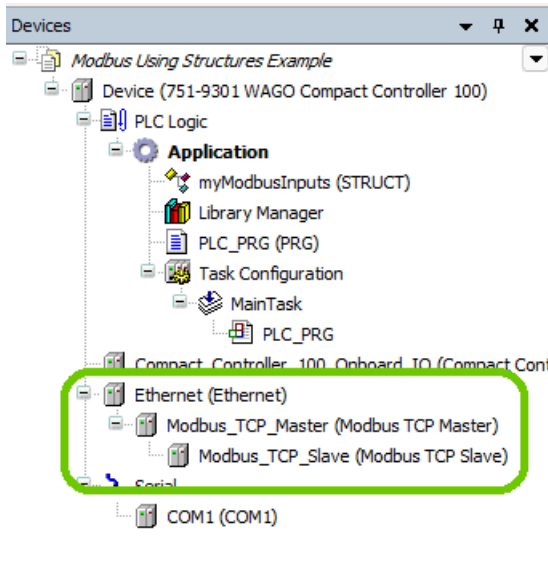
```
1 PROGRAM PLC_PRG
2 VAR
3     MB : MyModbusInputs;
4
```

Also declare a pointer to the structure type.

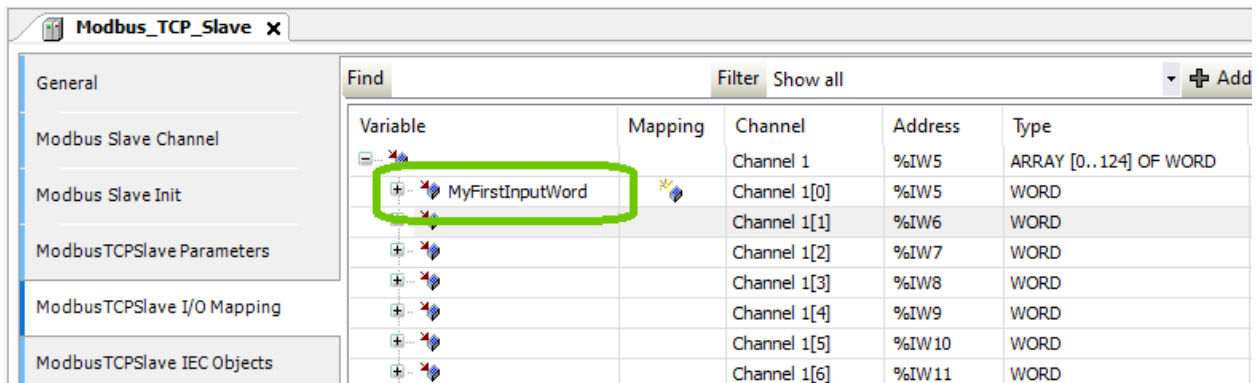


```
1 PROGRAM PLC_PRG
2 VAR
3     MB : MyModbusInputs;
4     pMB : POINTER TO myModbusInputs;
5
```

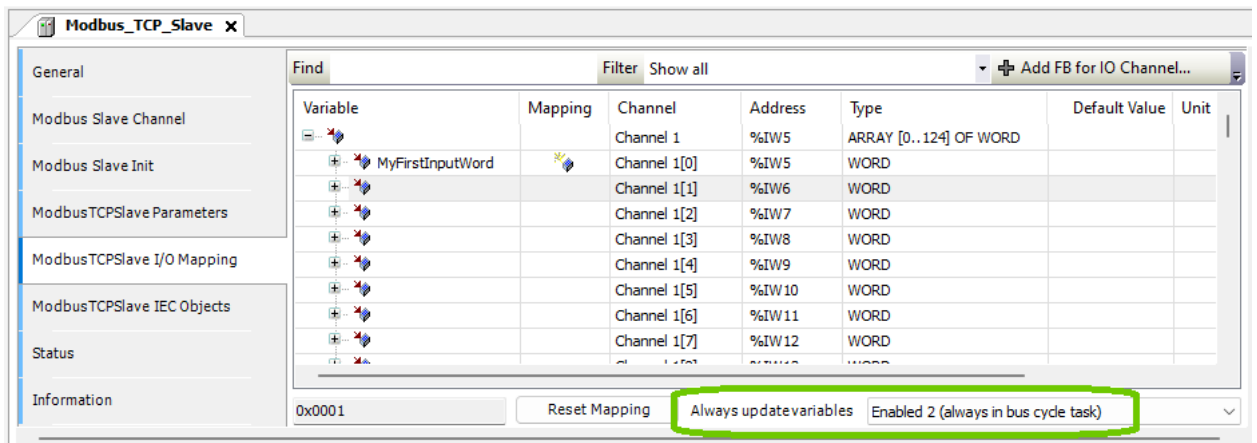
Add the necessary devices to define the Modbus TCP Slave connection.



Open the ModbusTCP Slave I/O Mapping editor, and type a variable name for the first word of input data. Here it is called 'MyFirstInputWord'.

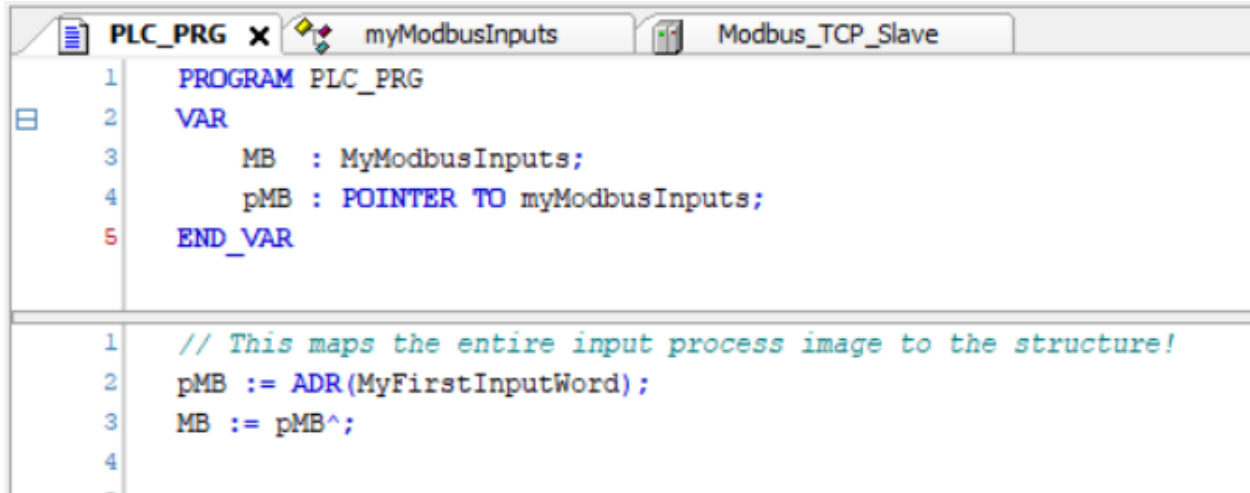


IMPORTANT: Make sure the 'Always update variables' parameter is enabled.



Now type two lines of code in your project that will map the entire Modbus input process image to the structure:

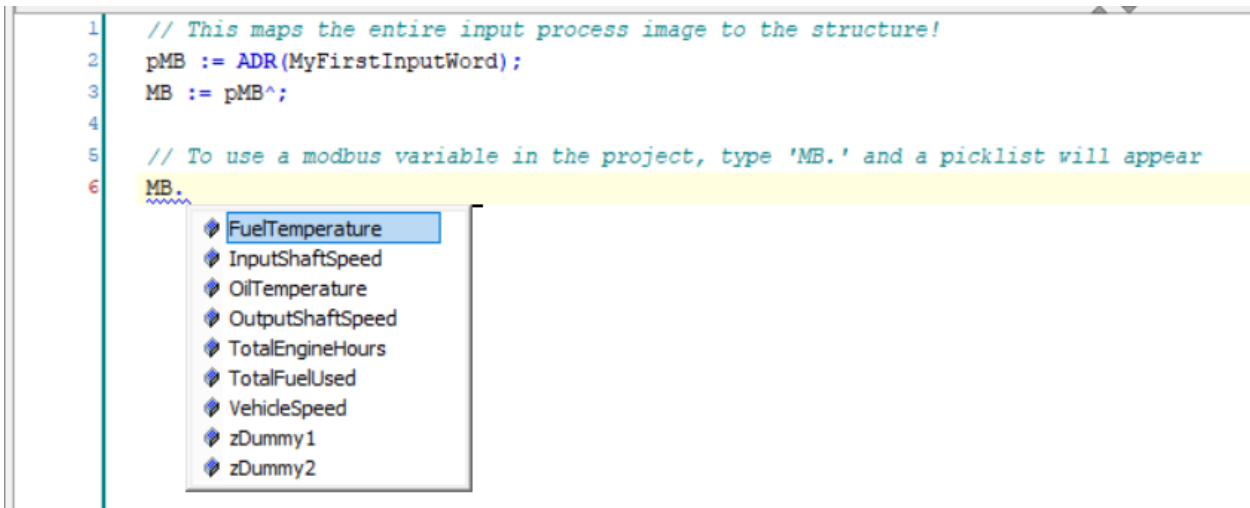
```
pMB := ADR(MyFirstInputWord);  
MB := pMB^;
```



The screenshot shows a PLC programming IDE with three tabs: 'PLC_PRG', 'myModbusInputs', and 'Modbus_TCP_Slave'. The 'PLC_PRG' tab is active and displays the following code:

```
1 PROGRAM PLC_PRG  
2 VAR  
3     MB : MyModbusInputs;  
4     pMB : POINTER TO myModbusInputs;  
5 END_VAR  
  
1 // This maps the entire input process image to the structure!  
2 pMB := ADR(MyFirstInputWord);  
3 MB := pMB^;  
4  
-
```

The entire input process image is now mapped, and you can start using the variables in the project. To do so, type 'MB.', and a picklist of each Modbus variable will appear for easy selection. Note that the variables appear alphabetically – this is why it is advantageous to start dummy variable names with the letter 'z'.



The screenshot shows the same PLC programming IDE with the following code:

```
1 // This maps the entire input process image to the structure!  
2 pMB := ADR(MyFirstInputWord);  
3 MB := pMB^;  
4  
5 // To use a modbus variable in the project, type 'MB.' and a picklist will appear  
6 MB.
```

A picklist menu is displayed below the 'MB.' text, listing the following variables alphabetically:

- FuelTemperature
- InputShaftSpeed
- OilTemperature
- OutputShaftSpeed
- TotalEngineHours
- TotalFuelUsed
- VehicleSpeed
- zDummy1
- zDummy2

Q. What if there are multiple slave devices?

A. If there are multiple slave devices, a unique structure should be declared for each device, and these two lines of code (with unique variable names) need to be added for each structure.

```
pMB := ADR(MyFirstInputWord);  
MB := pMB^;
```

Q. What if there is a change to the number of Modbus variables being communicated to a device?

A. This is one of the greatest advantages of this solution. If the Modbus variables change, just change the structure definition to match (using any editor). The variables will be remapped accordingly.