pbsSoftLogic Modbus Slave Driver Configuration

Feb 2026

Ver. 1.1

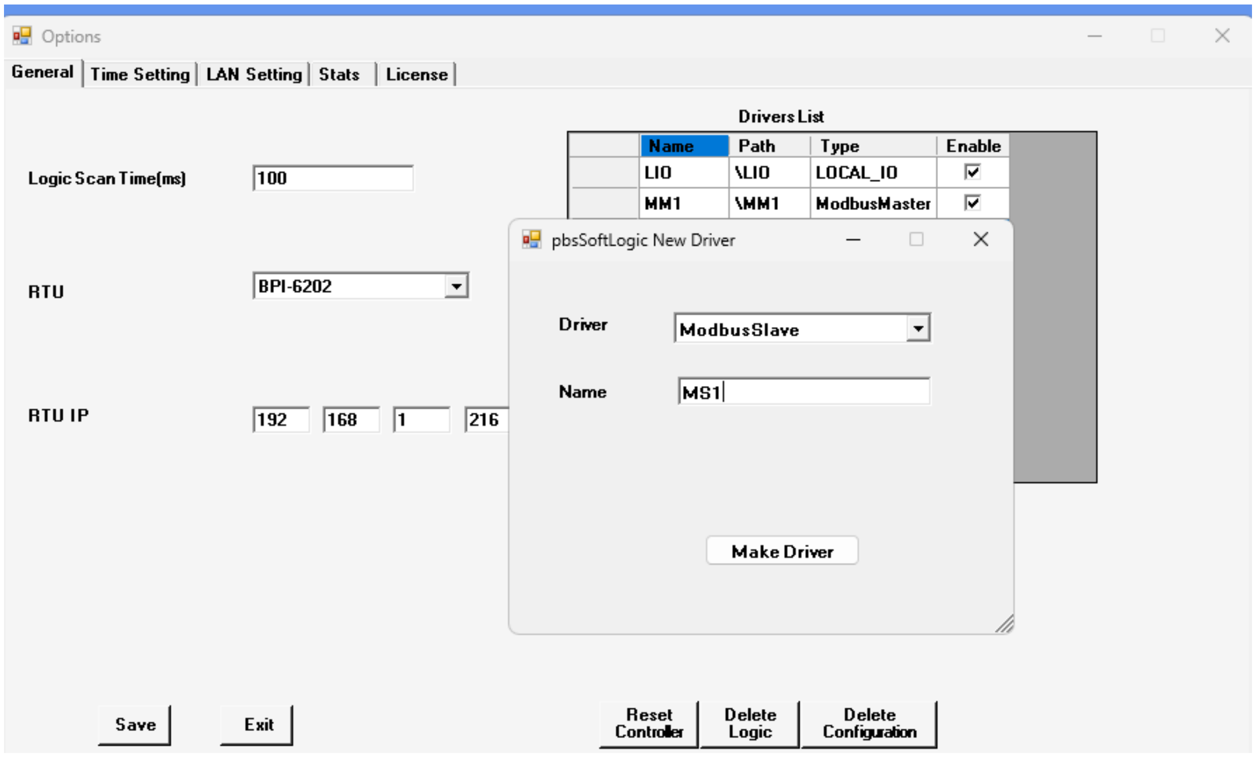
Kamjoo Bayat

Kb@pbscontrol.com

Introduction

The **Modbus Slave Driver** in the pbsSoftLogic platform provides full support for both **Modbus RTU** and **Modbus TCP** communication protocols. The driver is designed to be flexible and scalable, allowing each project to define and configure **multiple independent Modbus slave instances** as required. This enables seamless integration with a wide range of industrial devices and systems while maintaining consistent behavior across different communication modes.

To add a new **Modbus Slave Driver** to a project, open the *Project Options* page and right-click on the **Driver List** section. Select **Add New Driver** to create a new Modbus Slave instance. After the driver is added, click the **Make Driver** button to automatically generate the default configuration and associated tags for the project. Each driver must be assigned a **unique name** to ensure correct identification and avoid conflicts within the project.



Logic Scan Time(ms) 100

RTU BPI-6202

RTU IP 192 168 1 216

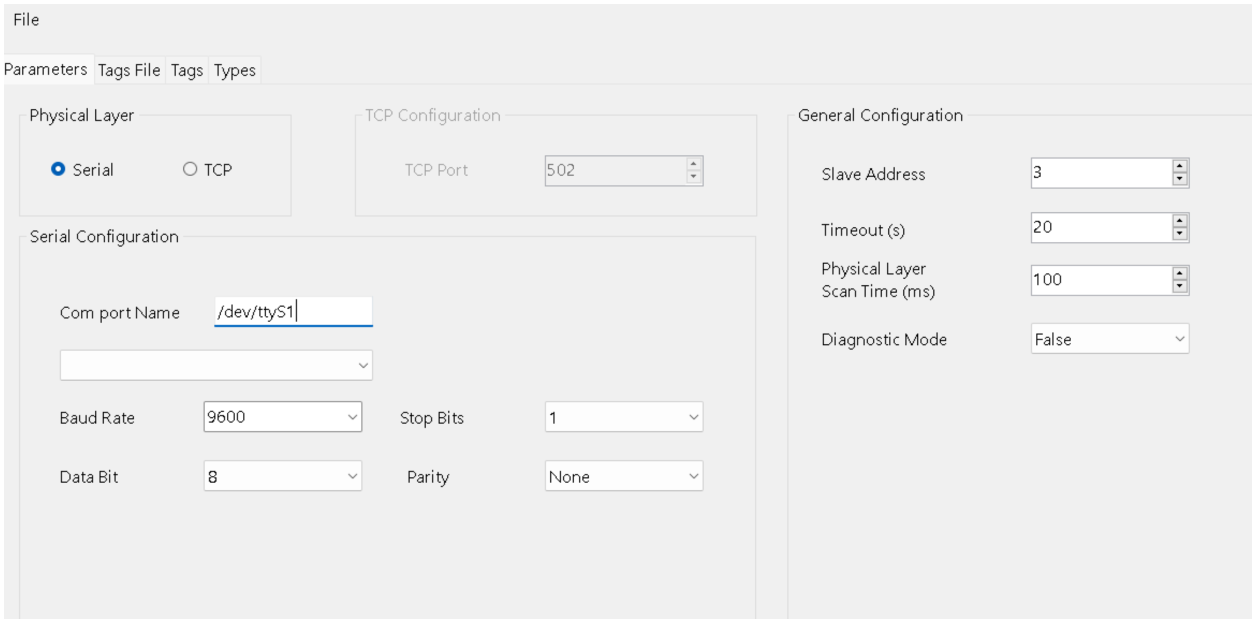
Drivers List

Name	Path	Type	Enable
LIO	\LIO	LOCAL_IO	✓
MM1	\MM1	ModbusMaster	✓

Driver ModbusSlave

Name MS1

Double-click the newly created Modbus Slave Driver to open its configuration page. The driver configuration interface will then be displayed.



Physical Layer: [✓ Serial] TCP

TCP Configuration
TCP Port: 502

Serial Configuration
Com port Name: /dev/ttyS1
Baud Rate: 9600
Stop Bits: 1
Data Bit: 8
Parity: None

General Configuration
Slave Address: 3
Timeout (s): 20
Physical Layer Scan Time (ms): 100
Diagnostic Mode: False

You can select either **Serial (Modbus RTU)** or **TCP (Modbus TCP)** as the physical communication layer for the Modbus Slave Driver.

Serial (Modbus RTU) Configuration

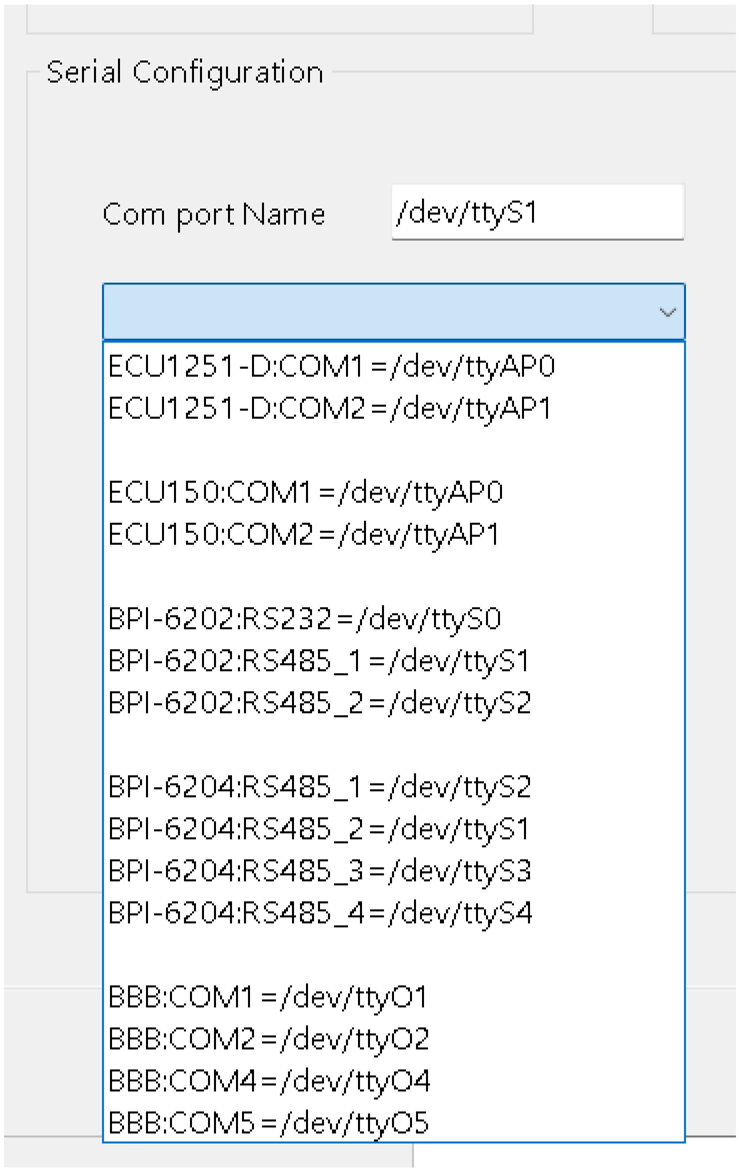
In the **Serial Configuration** section, you can define the following parameters:

- **COM Port Name**
- **Baud Rate**
- **Stop Bits**
- **Data Bits**
- **Parity**

The **COM Port Name** dropdown provides a list of typical serial port identifiers commonly used on well-known industrial hardware platforms.

TCP (Modbus TCP) Configuration

For **Modbus TCP Slave** mode, you can specify the **TCP Port** on which the driver will listen. If multiple Modbus TCP Slave drivers are defined within the same project, each instance must use a **different TCP port** (e.g., 502, 503, 504, …) to avoid port conflicts.


Serial Configuration

Com port Name /dev/ttyS1

```
ECU1251-D:COM1=/dev/ttyAP0
ECU1251-D:COM2=/dev/ttyAP1

ECU150:COM1=/dev/ttyAP0
ECU150:COM2=/dev/ttyAP1

BPI-6202:RS232=/dev/ttyS0
BPI-6202:RS485_1=/dev/ttyS1
BPI-6202:RS485_2=/dev/ttyS2

BPI-6204:RS485_1=/dev/ttyS2
BPI-6204:RS485_2=/dev/ttyS1
BPI-6204:RS485_3=/dev/ttyS3
BPI-6204:RS485_4=/dev/ttyS4

BBB:COM1=/dev/ttyO1
BBB:COM2=/dev/ttyO2
BBB:COM4=/dev/ttyO4
BBB:COM5=/dev/ttyO5
```

You cannot assign the **same serial port** to more than one Modbus Slave Driver. Each serial interface may be linked to **only a single driver instance**. For example, if **/dev/ttyS1** is already used by a Modbus Master Driver, it cannot be assigned to any other driver within the project.

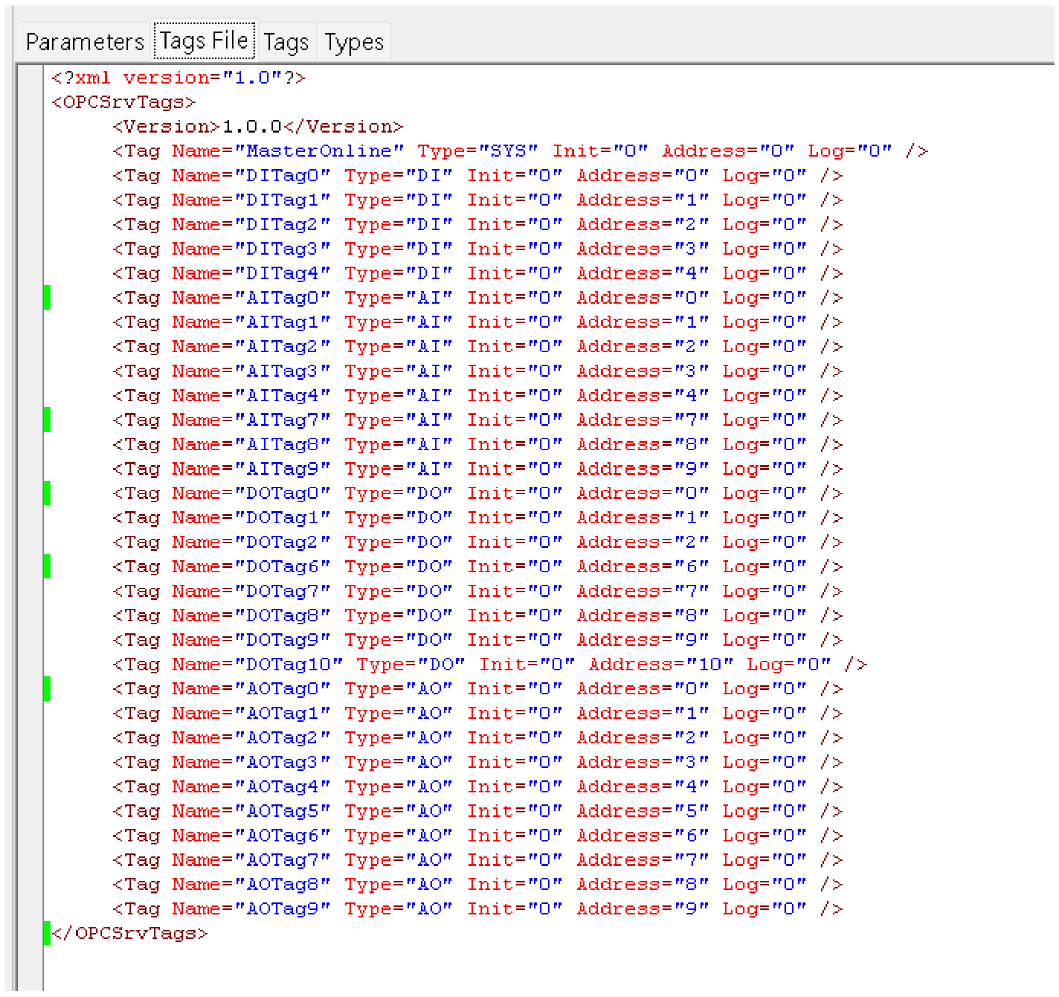
In the **General Configuration** section, you can define the **Slave Address**, **Timeout (sec)**, and **Diagnostic Mode** parameters.

- **Slave Address** specifies the Modbus address of the slave device. (1 to 255)
- **Timeout (sec)** determines how long the driver waits without receiving any request from a Modbus master. If no request is detected within the specified timeout period, the

driver automatically closes the Modbus TCP socket and reopens it, waiting for a new connection from a master device.

- **Diagnostic Mode** is disabled by default. When enabled, the driver outputs detailed Modbus frame information to the console. This option is useful when running the pbsSoftLogic runtime kernel manually on an RTU device for debugging or communication analysis.

To define **Modbus Slave Tags**, open the **Tags** page associated with the driver. When a new Modbus Slave Driver is created, pbsSoftLogic automatically generates a **default set of tags** to provide a ready-to-use starting point. These tags can be fully customized, modified, or expanded based on the specific requirements of your project.



```xml
<?xml version="1.0"?>
<OPCSrvTags>
     <Version>1.0.0</Version>
     <Tag Name="MasterOnline" Type="SYS" Init="0" Address="0" Log="0" />
     <Tag Name="DITag0" Type="DI" Init="0" Address="0" Log="0" />
     <Tag Name="DITag1" Type="DI" Init="0" Address="1" Log="0" />
     <Tag Name="DITag2" Type="DI" Init="0" Address="2" Log="0" />
     <Tag Name="DITag3" Type="DI" Init="0" Address="3" Log="0" />
     <Tag Name="DITag4" Type="DI" Init="0" Address="4" Log="0" />
     <Tag Name="AITag0" Type="AI" Init="0" Address="0" Log="0" />
     <Tag Name="AITag1" Type="AI" Init="0" Address="1" Log="0" />
     <Tag Name="AITag2" Type="AI" Init="0" Address="2" Log="0" />
     <Tag Name="AITag3" Type="AI" Init="0" Address="3" Log="0" />
     <Tag Name="AITag4" Type="AI" Init="0" Address="4" Log="0" />
     <Tag Name="AITag7" Type="AI" Init="0" Address="7" Log="0" />
     <Tag Name="AITag8" Type="AI" Init="0" Address="8" Log="0" />
     <Tag Name="AITag9" Type="AI" Init="0" Address="9" Log="0" />
     <Tag Name="DOTag0" Type="DO" Init="0" Address="0" Log="0" />
     <Tag Name="DOTag1" Type="DO" Init="0" Address="1" Log="0" />
     <Tag Name="DOTag2" Type="DO" Init="0" Address="2" Log="0" />
     <Tag Name="DOTag6" Type="DO" Init="0" Address="6" Log="0" />
     <Tag Name="DOTag7" Type="DO" Init="0" Address="7" Log="0" />
     <Tag Name="DOTag8" Type="DO" Init="0" Address="8" Log="0" />
     <Tag Name="DOTag9" Type="DO" Init="0" Address="9" Log="0" />
     <Tag Name="DOTag10" Type="DO" Init="0" Address="10" Log="0" />
     <Tag Name="AOTag0" Type="AO" Init="0" Address="0" Log="0" />
     <Tag Name="AOTag1" Type="AO" Init="0" Address="1" Log="0" />
     <Tag Name="AOTag2" Type="AO" Init="0" Address="2" Log="0" />
     <Tag Name="AOTag3" Type="AO" Init="0" Address="3" Log="0" />
     <Tag Name="AOTag4" Type="AO" Init="0" Address="4" Log="0" />
     <Tag Name="AOTag5" Type="AO" Init="0" Address="5" Log="0" />
     <Tag Name="AOTag6" Type="AO" Init="0" Address="6" Log="0" />
     <Tag Name="AOTag7" Type="AO" Init="0" Address="7" Log="0" />
     <Tag Name="AOTag8" Type="AO" Init="0" Address="8" Log="0" />
     <Tag Name="AOTag9" Type="AO" Init="0" Address="9" Log="0" />
</OPCSrvTags>
```

Each tag in the Modbus Slave Driver contains the following fields: **Name**, **Type**, **Init Value**, **Address**, and **Log**.

Name

A unique identifier for the tag within this driver instance.

Type

Specifies the Modbus data type and the corresponding function code. The supported types are:

Input Types

Code	Description	Function Code	Notes
DI	Digital Input Status	FC=02	Single-bit input
AI	Analog Input 16-bit Register	FC=04	16-bit register
FI	Floating-Point Input 32-bit	FC=04	Shares AI address space
INTI	Long Input 32-bit	FC=04	Shares AI address space
UINTI	Unsigned Long Input 32-bit	FC=04	Shares AI address space
SFI	Swapped Floating-Point Input 32-bit	FC=04	Shares AI address space
SINTI	Swapped Long Input 32-bit	FC=04	Shares AI address space
SUINTI	Swapped Unsigned Long Input 32-bit	FC=04	Shares AI address space

Output Types

Code	Description	Function Code	Notes
DO	Digital Output Coil	FC=05	Single-bit output
AO	Analog Output 16-bit Register	FC=06	16-bit register
FO	Floating-Point Output 32-bit	FC=16	Shares AO address space
INTO	Long Output 32-bit	FC=16	Shares AO address space
UINTO	Unsigned Long Output 32-bit	FC=16	Shares AO address space
SFO	Swapped Floating-Point Output 32-bit	FC=16	Shares AO address space

SINTO	Swapped Long Output 32-bit	FC=16	Shares AO address space
SUINTO	Swapped Unsigned Long Output 32-bit	FC=16	Shares AO address space

Init Value

Defines the initial value of the tag. When the RTU device restarts, the Modbus Slave Driver loads this value before communication begins.

Address

Specifies the **starting Modbus register address** for the tag.

For multi-register data types (such as FI, FO, INTI, etc.), the address must account for the number of registers consumed.

Example: If you define two **FI (32-bit float)** tags, they occupy registers:

- Tag 1 → registers **10–11**
- Tag 2 → registers **12–13**

Therefore:

- Tag 1 **Address = 10**
- Tag 2 **Address = 12**

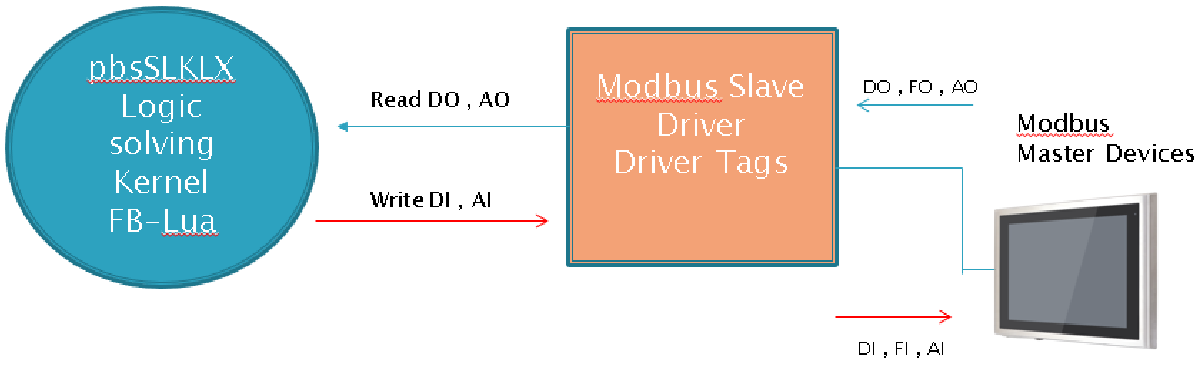
This ensures proper alignment and prevents register overlap.

All **input tags** (DI, AI, FI, INTI, …) must be **written and updated by your SoftLogic program**. When a Modbus master reads these tags, it receives the values that your logic has previously written into them.

All **output tags** (DO, AO, FO, INTO, …) are **read by your logic**. When a Modbus master writes to these tags, the Modbus Slave Driver updates their values, and your SoftLogic program must read and use them accordingly.

Example: If a Modbus master reads **DI**, **AI**, or **FI** tags, your logic must continuously write updated values to these tags. If the master writes to **DO**, **AO**, or **FO** tags, your logic must read the updated tag values and apply them inside the control program.

This separation ensures a clear and consistent data flow between the Modbus master and the SoftLogic runtime.



All **analog input tags** (AI, FI, INTI, UINTI, SFI, SINTI, …) share the **same input-register address space**. Therefore, two different input tags cannot use the **same starting address**. For example, you cannot define an **FI** tag at address **32** and also define an **INTI** tag starting at address **32**. Each tag must occupy its own non-overlapping register range.

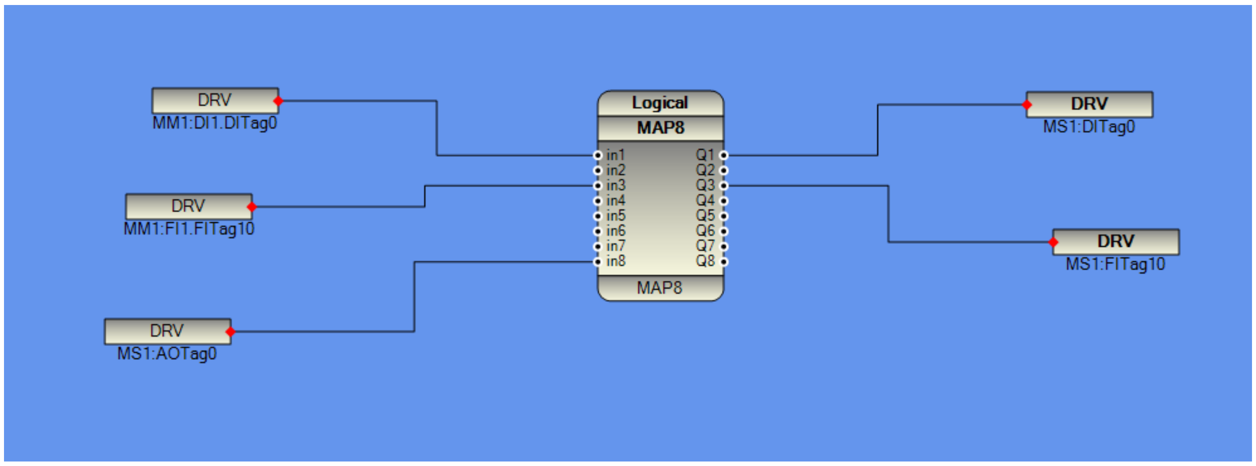
The **same rule applies to analog output tags**. All output types (AO, FO, INTO, UINTO, SFO, SINTO, …) share a common output-register address space, and their register ranges must not overlap.

```
  <Tag Name="AITag0" Type="AI" Init="0" Address="0" Log="0" />
  <Tag Name="AITag1" Type="AI" Init="0" Address="1" Log="0" />
  <Tag Name="AITag2" Type="AI" Init="0" Address="2" Log="0" />
  <Tag Name="AITag3" Type="AI" Init="0" Address="3" Log="0" />
  <Tag Name="AITag4" Type="AI" Init="0" Address="4" Log="0" />
  <Tag Name="AITag7" Type="AI" Init="0" Address="7" Log="0" />
  <Tag Name="AITag8" Type="AI" Init="0" Address="8" Log="0" />
  <Tag Name="AITag9" Type="AI" Init="0" Address="9" Log="0" />

 <Tag Name="FITag10" Type="FI" Init="0" Address="10" Log="0" />
 <Tag Name="FITag12" Type="FI" Init="0" Address="12" Log="0" />

 <Tag Name="LITag14" Type="INTI" Init="0" Address="14" Log="0" />
```

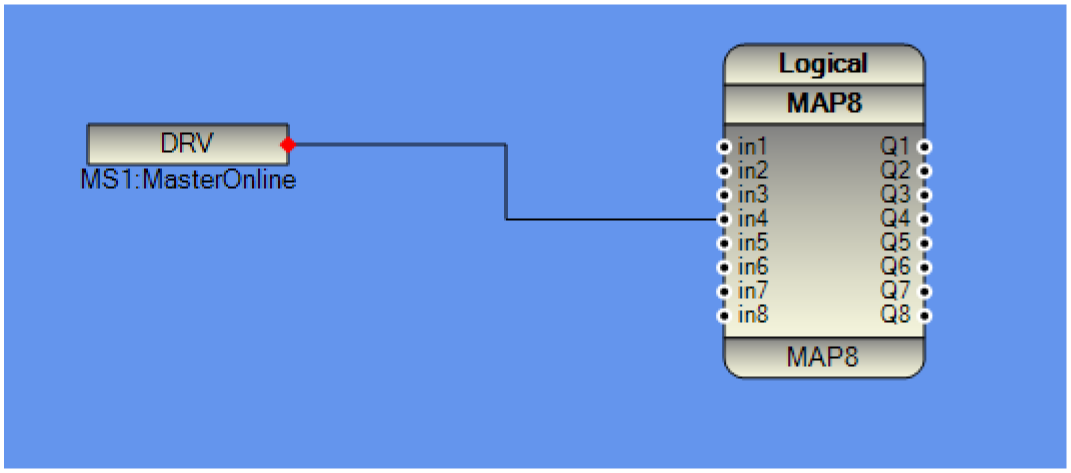
Consider the following simple logic example. On the **left side** of the Map8 Function Block, values are **read** from the Modbus Master Driver (MM1) and then mapped to two tags belonging to the Modbus Slave Driver. As a general rule in pbsSoftLogic:

- **Left side of a function block = reading tag values**
- **Right side of a function block = writing to tags**

In this example, the logic reads data from the Modbus Master and writes the mapped values into the corresponding Modbus Slave tags.



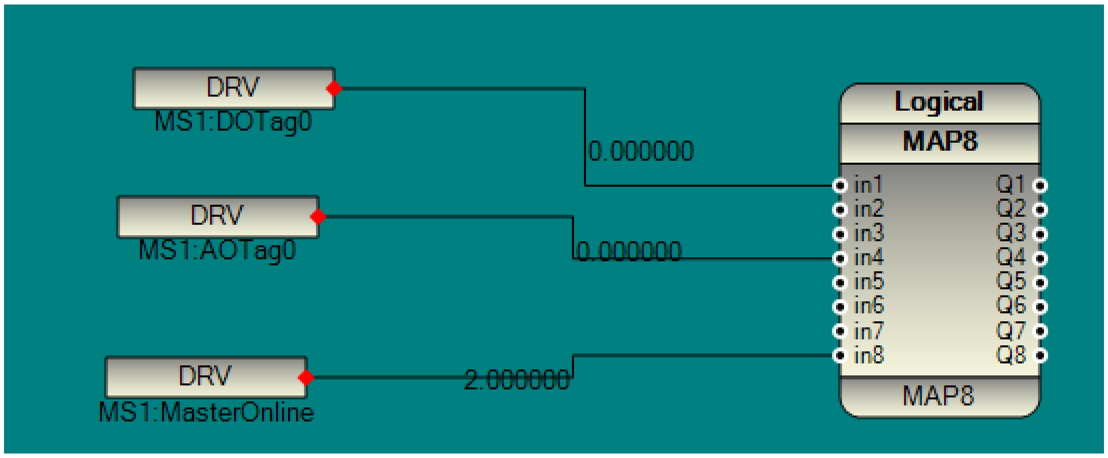
The system tag **MasterOnline** indicates whether a Modbus master device is currently connected and communicating with the driver. When the tag is **True**, one master is online; when **False**, no master is connected.



pbsSoftLogic Modbus Slave Driver fully supports **multiple simultaneous Modbus TCP masters**. The system tag MasterOnline indicates the **current number of active master connections**.

Whenever a new Modbus TCP master establishes a connection with the RTU, the value of MasterOnline increases by one. When a master disconnects, the value decreases accordingly.

In the example below, **two Modbus TCP masters** are connected to the RTU at the same time.


DRV
MS1:DOTag0
0.000000
DRV
MS1:AOTag0
0.000000
DRV
MS1:MasterOnline
2.000000
Logical
MAP8
in1
in2
in3
in4
in5
in6
in7
in8
Q1
Q2
Q3
Q4
Q5
Q6
Q7
Q8
MAP8