MODBU Communication Driver

Driver for Serial Communication
with Devices Using the Modbus Protocol

Contents

Introduction

The MODBU driver enables serial communication between the Studio system and remote devices using the Modbus protocol, according to the specifications discussed in this document.

This document will help you to select, configure and execute the MODBU driver, and it is organized as follows:

- **Introduction**: This section, which provides an overview of the document.
- **General Information**: Identifies all of the hardware and software components required to implement communication between the Studio system and the target device.
- **Selecting the Driver**: Explains how to select the MODBU driver in the Studio system.
- **Configuring the Device**: Describes how the target device must be configured to receive communication from the MODBU driver.
- **Configuring the Driver**: Explains how to configure the MODBU driver in the Studio system, including how to associate database tags with device registers.
- **Executing the Driver**: Explains how to execute the MODBU driver during application runtime.
- **Troubleshooting**: Lists the most common errors for this driver, their probable causes, and basic procedures to resolve them.
- **Sample Application**: Explains how to use a sample application to test the MODBU driver configuration
- **Revision History**: Provides a log of all changes made to the driver and this documentation.

Notes:
- This document assumes that you have read the “Development Environment” chapter in Studio’s *Technical Reference Manual*.
- This document also assumes that you are familiar with the Microsoft Windows XP/Vista/7 environment. If you are not familiar with Windows, then we suggest using the **Help** feature (available from the Windows desktop **Start** menu) as you work through this guide.

General Information

This chapter identifies all of the hardware and software components required to implement serial communication between the MODBU driver in Studio and remote devices using the Modbus protocol.

The information is organized into the following sections:

- Device Specifications
- Network Specifications
- Driver Characteristics
- Conformance Testing

Device Specifications

To establish serial communication, your target device must meet the following specifications:

- **Manufacturer:** Any device using the Modbus protocol for serial communication
- **Compatible Equipment:** Any device that is compatible with the Modbus protocol
- **Programmer Software:** It depends on the device

Network Specifications

To establish communication, your device network must meet the following specifications:

- **Device Communication Port**: Modbus Serial Port
- **Physical Protocol**: RS232 / RS845
- **Logic Protocol**: Modbus
- **Device Runtime Software**: None
- **Specific PC Board**: None
- **Cable Wiring Scheme**: it depends on the device

Driver Characteristics

The MODBU driver package consists of the following files, which are automatically installed in the **`\DRV`** subdirectory of Studio:

- **`MODBU.INI:`** Internal driver file. *You must not modify this file*.
- **`MODBU.MSG:`** Internal driver file containing error messages for each error code. *You must not modify this file*.
- **`MODBU.PDF:`** This document, which provides detailed information about the MODBU driver.
- **`MODBU.DLL:`** Compiled driver.

> **Note:**
> You must use Adobe Acrobat® Reader™ to view the **`MODBU.PDF`** document. You can install Acrobat Reader from the Studio installation CD, or you can download it from Adobe's Web site.

You can use the MODBU driver on the following operating systems:

- Windows 7/8
- Windows Embedded and CE 5.x, 6.x, 7.x

For a description of the operating systems used to test driver conformance, see “Conformance Testing” below.

The MODBU driver supports the following registers:

Register Type	Length	Write	Read	Bit	Integer	Float	DWord	BCD	BCD DW	STRING
0x (Coil Status)	1 Bit	•	•	•	–	–	–	–	–	–
1x (Input Status)	1 Bit	–	•	•	–	–	–	–	–	–
3x (Input Register)	1 Word	–	•	•	•	•	•	•	•	–
4x (Holding Register)	1 Word	•	•	•	•	•	•	•	•	•
6x (File Records)	n Word	–	•	–	–	–	–	–	–	–

Conformance Testing

The following hardware/software was used for conformance testing:

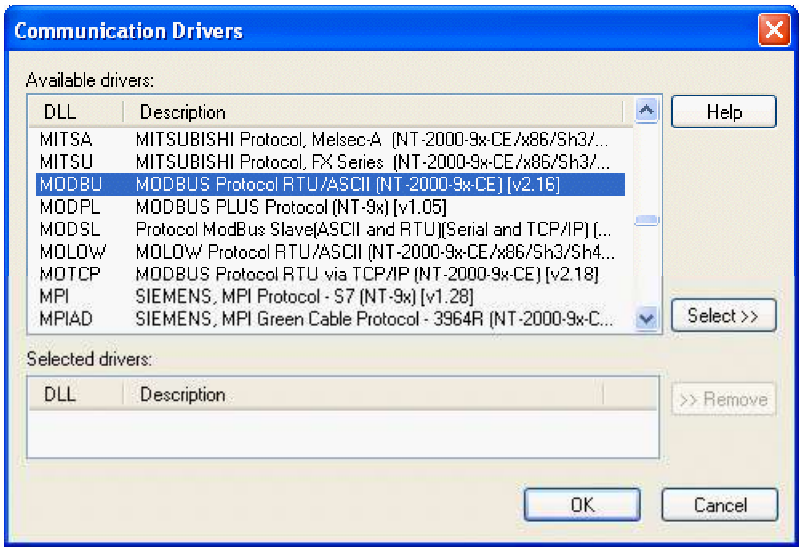
- **Configuration**:
  – **Modbus Port**: 1
  – **Baud Rate**: 9600
  – **Protocol**: RTU
  – **Data Bits**: 8
  – **Stop Bits**: 1
  – **Parity**: Odd
  – **COM Port**: COM6
- **Cable**: Use specifications described in the “Network Specifications” section above.

Driver Version	Studio Version	Operating System (development)	Operating System (target)	Equipment
10.12	8.0 + SP2	Windows 8	Windows 7/8 Windows CE 6.0 Ubuntu 14.04 LTS Raspbian Jessie	GE-Fanuc 90-30 Model Modbus Simulator Automation Direct’s DL 205

Selecting the Driver

When you install Studio, all of the communication drivers are automatically installed in the **\DRV** subdirectory but they remain dormant until manually selected for specific applications. To select the MODBU driver for your Studio application:

1. From the main menu bar, select **Insert** → **Driver** to open the *Communication Drivers* dialog.
2. Select the **MODBU** driver from the *Available Drivers* list, and then click the **Select** button.



DLL	Description
MITSA	MITSUBISHI Protocol, Melsec-A (NT-2000-9x-CE/x86/Sh3/...
MITSU	MITSUBISHI Protocol, FX Series (NT-2000-9x-CE/x86/Sh3/...
MODBU	MODBUS Protocol RTU/ASCII (NT-2000-9x-CE) [v2.16]
MODPL	MODBUS PLUS Protocol (NT-9x) [v1.05]
MODSL	Protocol ModBus Slave(ASCII and RTU)(Serial and TCP/IP) (...
MOLOW	MOLOW Protocol RTU/ASCII (NT-2000-9x-CE/x86/Sh3/Sh4...
MOTCP	MODBUS Protocol RTU via TCP/IP (NT-2000-9x-CE) [v2.18]
MPI	SIEMENS, MPI Protocol - S7 (NT-9x) [v1.28]
MPIAD	SIEMENS, MPI Green Cable Protocol - 3964R (NT-2000-9x-C...

DLL	Description

Communication Drivers Dialog

3. When the **MODBU** driver is displayed in the **Selected Drivers** list, click the **OK** button to close the dialog. The driver is added to the *Drivers* folder, in the *Comm* tab of the Workspace.

➲ **Attention:**
For safety reasons, you must take special precautions when installing any physical hardware. Please consult the manufacturer’s documentation for specific instructions.

Configuring the Device

Because there are several brands of equipment that use the Modbus protocol, it is impossible to define a standard device configuration. Therefore, we suggest using the following default configuration:

- **Protocol**: RTU
- **Baud Rate**: 9600
- **Data Bits**: 8
- **Stop Bits**: 1
- **Parity**: None

Configuring the Driver

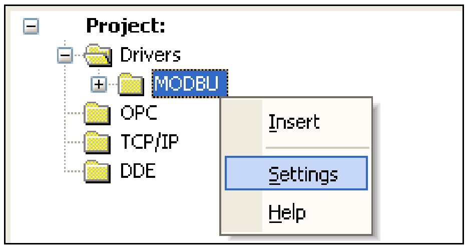
Once you have selected the MODBU driver in Studio, you must properly configure it to communicate with your target device. First, you must set the driver's communication settings to match the parameters set on the device. Then, you must build driver worksheets to associate database tags in your Studio application with the appropriate addresses (registers) on the device.

Configuring the Communication Settings

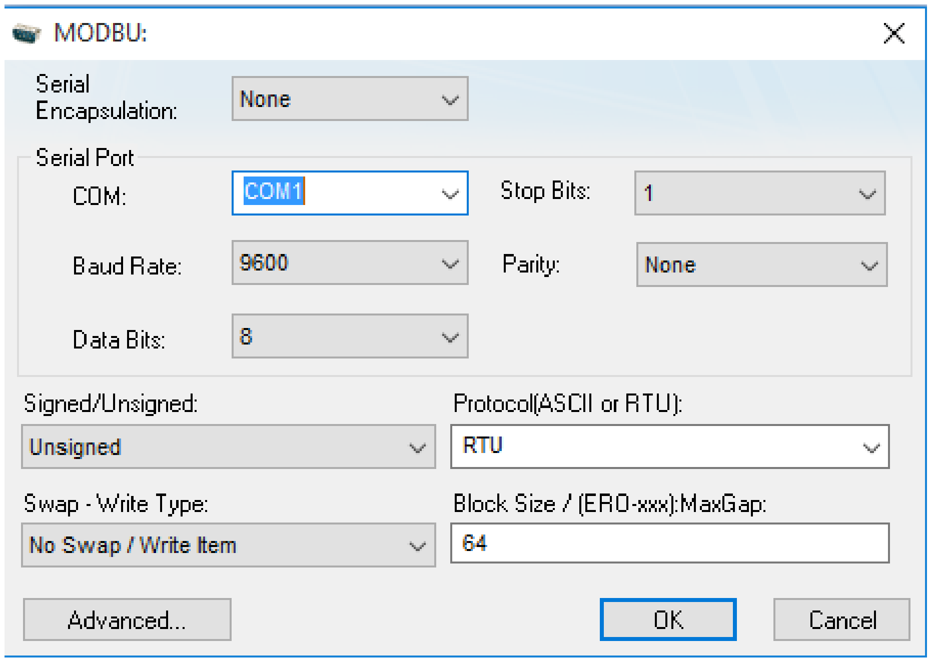
The communication settings are described in detail in the "Communication" chapter of the Studio *Technical Reference Manual*, and the same general procedures are used for all drivers. Please review those procedures before continuing.

For the purposes of this document, only MODBU driver-specific settings and procedures will be discussed here. To configure the communication settings for the MODBU driver:

1. In the *Workspace* pane, select the *Comm* tab and then expand the *Drivers* folder. The MODBU driver is listed here as a subfolder.
2. Right-click on the *MODBU* subfolder and then select the **Settings** option from the pop-up menu. The *MODBU: Communication Parameters* dialog is displayed:



Select Settings from the Pop-Up Menu


MODBU:

Serial Encapsulation: None

Serial Port
COM: COM1
Stop Bits: 1
Baud Rate: 9600
Parity: None
Data Bits: 8

Signed/Unsigned: Unsigned
Protocol(ASCII or RTU): RTU
Swap - Write Type: No Swap / Write Item
Block Size / (ERO-xxx):MaxGap: 64

MODBU: Communication Parameters Dialog

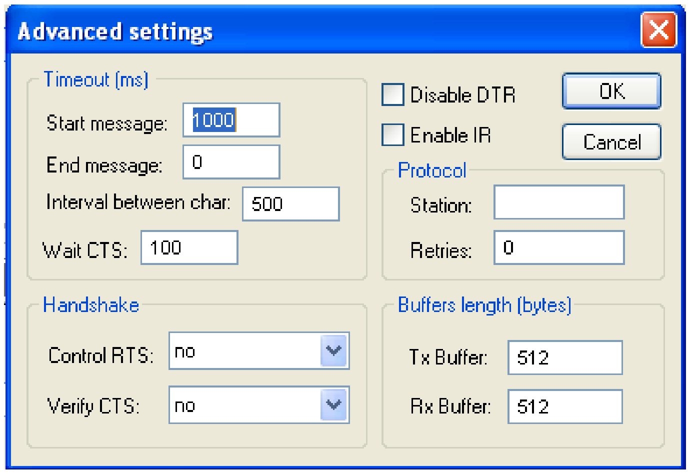
3. Verify the *Serial Port* settings, and change them if necessary.
4. Configure the additional driver-specific settings, as described in the following table:

Setting	Default Value	Valid Values	Description	
Serial Port	`COM 1`	`COM1`	COM1 etc for using the driver on Windows	See Appendix A for configuration and running the driver on Linux devices
		`/dev/ttyS1`	When using serial ports on linux devices using IotView	
		`/dev/ttyUSB0`	when using USB to Serial convertors for example on Raspberry Pi devices	
Signed / Unsigned	`Unsigned`	`Unsigned(legacy)`	You should not use these options because they are just to keep compatibility with older versions.	Note: This field just works using **3X**, **4X**, **DW**, **DWS**, **DW3**, **DW3S** or **ID** register types.
		`Signed(legacy)`		
		`Unsigned`	No allow negative values. INTEGER ➔ 0 to 65535. DWORD ➔ 0 to 4294967295.	
		`Signed`	Allow negative values.	

			INTEGER ➔ -32768 to 32767. DWORD ➔ -2147483648 to 2147483647.	
Protocol (ASCII or RTU)	`RTU`	`ASCII`	Each eight-bit Word is sent as two four-bit ASCII characters, allowing for a time interval between characters without causing errors.	
		`RTU`	Each eight-bit Word is sent as two four-bit hexadecimals, allowing for greater density and faster throughput. NOTE: In most cases, we recommend using this protocol.	
Swap – Write Type	`No Swap / Write Item`	`No Swap or Swap`	This option enables or disables the Swap Word order in the **reading** and **writing** commands. **No Swap** – disable Swap Word order **Swap** – enable Swap Word order	
			Note: -This parameter just will work with the **FP**, **FPS**, **FP3**, **FP3S**, **DW**, **DWS**, **DW3**, **DW3S**, **DF, DF3, DFS and DF3S** register types. **-** This parameter will affect **DF**, **DF3**, **DFS** and **DF3S** in a different way, converting their bytes order from Big-Endian to Little-Endian. (example in page 15)	
		`Write Item or Write Group`	This option chooses the function 0x06 or 0x10 to write. **Write Item** - Write on tag change (Standard Driver Sheet) and Write (Main Driver Sheet) use the Modbus function 0x06. - Write Trigger (Standard Driver Sheet) uses the Modbus function 0x10. **Write Group** – Any writing in the driver uses the Modbus function 0x10.	
			Note: - Modbus function **0x06** (Preset Single Register) - Modbus function **0x10** (Preset Multiple Registers) For more information about the functions consult the protocol guide	
Block Size / (ERO-xxx)	**64**	`0` to `512`	The nominal size (in Words) of each block of data to be transmitted, as determined by the processing capacity of the device. NOTE: A block size of 1 configured will send 1 item.	
		`ERO-`***`xxx`***	Address used to set equipment to Local or Remote (used for ERO equipment *only*).	
		`0` to `512` (Max Gap)	This parameter is used with Main Driver Sheet and configures the Gap size between 2 addresses that could belong to the same group or block according to the Block Size but, if there are no other I/O addresses between them, and their addresses difference is bigger than the GAP, they will end up in separated virtual read groups. This parameter can never be higher than the **Block Size**.	

Note:
It is not possible using **0X**, **1X**, **STA**, **FP**, **FP3**, **FP3S**, **FPS**, **BCD**, **BCD3**, **BCDDW**, **BCDDWS**, **BCDDW3**, **BCDDW3S**, **ST**, **STS, STU or STUS** register types with **Signed / Unsigned.** This field does not work using any one of them.

5. If you are using a Data Communication Equipment (DCE) converter (e.g., 232/485) between your PC and your target device, then you must also adjust the **Control RTS** (Request to Send) setting to account for the converter. In the *Communication Settings* dialog, click the **Advanced** button to open the *Advanced Settings* dialog:



Timeout (ms)
Start message: 1000
End message: 0
Interval between char: 500
Wait CTS: 100
Disable DTR
Enable IR
Protocol
Station:
Retries: 0
Handshake
Control RTS: no
Verify CTS: no
Buffers length (bytes)
Tx Buffer: 512
Rx Buffer: 512

Advanced Settings Dialog

When the dialog is displayed, configure the Control RTS setting using the following information:

Setting	Default	Values	Description
Control RTS	no	no	Do not set the RTS (Request to Send) handshake signal.
		Yes	Set the RTS (Request to Send) handshake signal before communication.
		Yes+echo	Set the RTS (Request to Send) handshake signal before communication, and echo the signal received from the target device.
		Always On	Set the RTS (Request to Send) handshake and keep it on

Attention:
If you incorrectly configure the **Control RTS** setting, then runtime communication will fail and the driver will generate a –15 error. See "Troubleshooting" for more information.

You do not need to change any other advanced settings at this time. You can consult the Studio *Technical Reference Manual* later for more information about configuring these settings.

6. Click **OK** to close the *Advanced Settings* dialog, and then click **OK** to close the *Communication Settings* dialog.

Configuring the Driver Worksheets

Each selected driver includes a Main Driver Sheet and one or more Standard Driver Worksheets. The Main Driver Sheet is used to define tag/register associations and driver parameters that are in effect at all times, regardless of application behavior. In contrast, Standard Driver Worksheets can be inserted to define additional tag/register associations that are triggered by specific application behaviors.

The configuration of these worksheets is described in detail in the “Communication” chapter of the Studio *Technical Reference Manual*, and the same general procedures are used for all drivers. Please review those procedures before continuing.

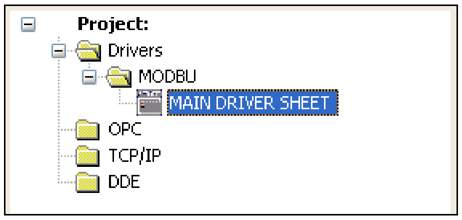
For the purposes of this document, only MODBU driver-specific parameters and procedures are discussed here.

MAIN DRIVER SHEET

When you select the MODBU driver and add it to your application, Studio automatically inserts the *Main Driver Sheet* in the *MODBU* driver subfolder. To configure the Main Driver Sheet:

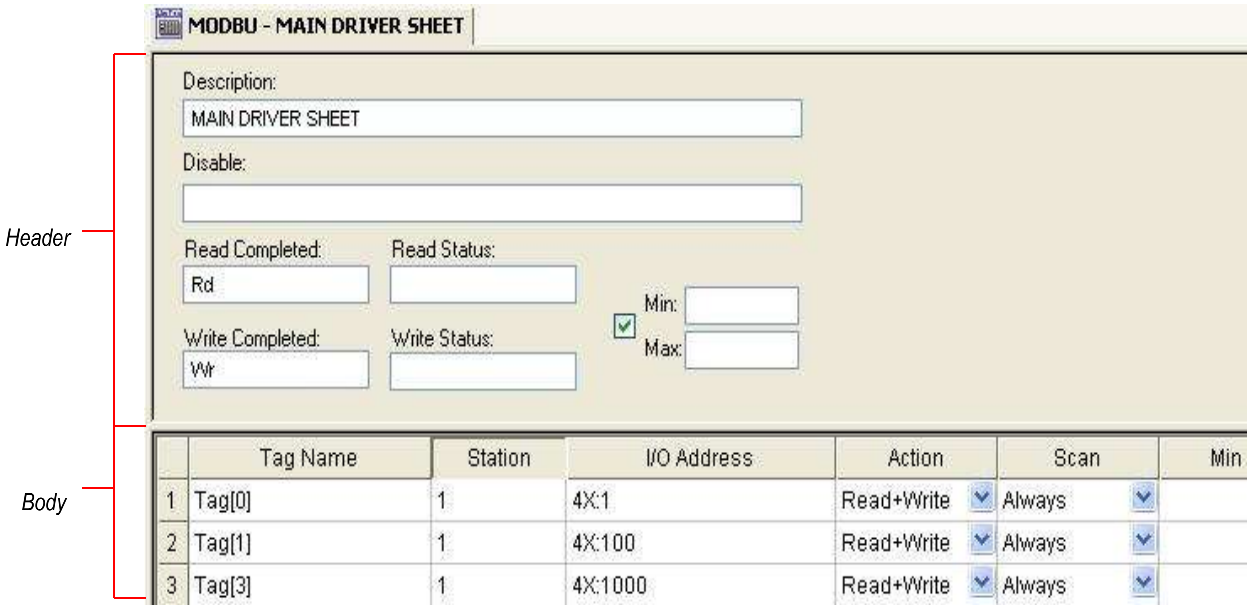
1. Select the *Comm* tab in the *Workspace* pane.

2. Open the *Drivers* folder, and then open the *MODBU* subfolder:



Main Driver Sheet in the MODBU Subfolder

3. Double-click on the **MAIN DRIVER SHEET** icon to open the following worksheet:



Description: MAIN DRIVER SHEET
Disable:
Read Completed: Rd
Read Status:
Write Completed: Wr
Write Status:
✓ Min:
Max:

	Tag Name	Station	I/O Address	Action	Scan	Min
1	Tag[0]	1	4X:1	Read+Write	Always	
2	Tag[1]	1	4X:100	Read+Write	Always	
3	Tag[3]	1	4X:1000	Read+Write	Always	

Opening the Main Driver Sheet

Most of the fields on this sheet are standard for all drivers; see the “Communication” chapter of the *Technical Reference Manual* for more information on configuring these fields. However, the **Station** and **I/O Address** fields use syntax that is specific to the MODBU driver.

4. For each table row (i.e., each tag/register association), configure the **Station** and **I/O Address** fields as follows:

   - **Station** field: Specify the ID number (node) of the target Modbus device, using the following syntax:

     <ID Number>

     Example — **57**

     Where ***<ID Number>*** is a value between 0 and 247.

     You can also specify an indirect tag (e.g. **{station}**), but the tag that is referenced must follow the same syntax and contain a valid value.

     > ➲ **Attention:**
     > You cannot leave the **Station** field blank.

   - **I/O Address:** Specify the address of the associated device register.

     For all register types other than **ST**/**STS** (String Type), use the following syntax:

     <Type>*:*[Signed/Unsigned]<Address>*.*[Bit]

     Examples — **4X:20** , **4X:S15** , **4X:10.7**

     For **ST/STS (String), STU/STUS** (Unicode String) registers *only*, use the following syntax:

     <Type>*:*<Address>*:*<Length>

     Example — **ST:10:5**

     Where:

     – ***<Type>*** : Register type. Valid values are **0X**, **1X**, **STA**, **3X**, **4X**, **FP**, **FP3**, **FP3S**, **DW**, **FPS**, **DWS**, **DW3**, **DW3S**, **BCD**, **BCD3**, **BCDDW**, **BCDDWS**, **BCDDW3**, **BCDDW3S**, **ID**, **ST**, **STS**, **DF**, **DF3**, **DFS**, **DF3S**, **STU and STUS**.

     – ***[Signed/Unsigned]*** (optional): Parameter used for integer values only. Valid values are **S** (Signed) and **U** (Unsigned). If you do not specify this parameter, then Studio uses the default parameter in the *Communication Settings* dialog.

     – ***<Address>*** : Address of the device register.

     – ***[Bit]*** (optional): Use this parameter only for **3X** (Input Register) and **4X** (Holding Register) types, to indicate which bit on the register will be read from and/or written to.

     – ***<Length>*** : Length of the string (in bytes) to be read or written.

     > ➲ **Attention:**
     > - For DWord registers (DW, DWS, DW3 and DW3S) using the unsigned option in the Address field, the associated database tag must be Real Type.
     > - The Floating-point values are 4 bytes using 6 significant digits.
     > - The Double Floating-point values (DF, DF3) are 8 bytes using 16 significant digits.
     > - Main Driver Sheet does not support the Read/Write of File Records.

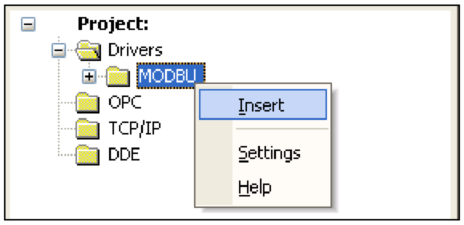
STANDARD DRIVER WORKSHEET

When you select the MODBU driver and add it to your application, it has only a Main Driver Sheet by default (see previous section). However, you may insert additional Standard Driver Worksheets to define tag/register associations that are triggered by specific application behaviors. Doing this will optimize communication and improve system performance by ensuring that tags/registers are scanned only when necessary – that is, only when the application is performing an action that requires reading or writing to those specific tags/registers.

Note:
We recommend configuring device registers in sequential blocks in order to maximize performance.

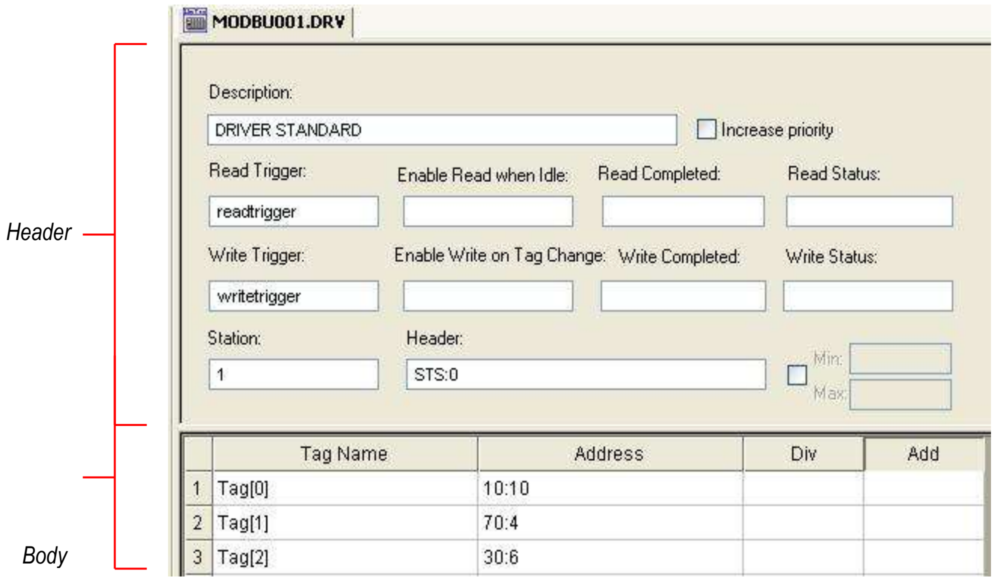
To insert a new Standard Driver Worksheet:

1. In the *Comm* tab, open the *Drivers* folder and locate the *MODBU* subfolder.
2. Right-click on the *MODBU* subfolder, and then select **Insert** from the pop-up menu:



Inserting a New Worksheet

A new MODBU driver worksheet is inserted into the *MODBU* subfolder, and the worksheet is opened for configuration:



Description: DRIVER STANDARD
Increase priority: ☐
Read Trigger: readtrigger
Enable Read when Idle:
Read Completed:
Read Status:
Write Trigger: writetrigger
Enable Write on Tag Change:
Write Completed:
Write Status:
Station: 1
Header: STS:0
Min:
Max:

	Tag Name	Address	Div	Add
1	Tag[0]	10:10		
2	Tag[1]	70:4		
3	Tag[2]	30:6		

MODBU Driver Worksheet

> **Note:**
> Worksheets are numbered in order of creation, so the first worksheet is **`MODBU001.drv`**.

Most of the fields on this worksheet are standard for all drivers; see the "Communication" chapter of the *Technical Reference Manual* for more information on configuring these fields. However, the **Station**, **Header**, and **Address** fields use syntax that is specific to the MODBU driver.

3. Configure the **Station** and **Header** fields as follows:

   - **Station** field: Specify the ID number (node) of the target Modbus device, using the following syntax:

     `<ID Number>`

     Example — **`57`**

     Where ***`<ID Number>`*** is a value between 0 and 247.

     You can also specify an indirect tag (e.g. **`{station}`**), but the tag that is referenced must follow the same syntax and contain a valid value.

     > **Attention:**
     > You cannot leave the **Station** field blank.

   - **Header** field: Specify the address of the first register of a block of registers on the target device. The addresses declared in the *Body* of the worksheet are simply offsets of this **Header** address. When Read/Write operations are executed for the entire worksheet (see **Read Trigger** and **Write Trigger** above), it scans the entire block of registers from the first address to the last.

     The **Header** field uses the following syntax:

     `<Type>`**`:`*****`<AddressReference>`***

     Example — **`4X:10`**

     Where:

     – ***`<Type>`*** is the register type (**`0X`**, **`1X`**, **`STA`**, **`3X`**, **`4X`**, **`FP`**, **`FP3`**, **`FP3S`**, **`DW`**, **`FPS`**, **`DWS`**, **`DW3`**, **`DW3S`**, **`BCD`**, **`BCD3`**, **`BCDDW`**, **`BCDDWS`**, **`BCDDW3`**, **`BCDDW3S`**, **`ID`**, **`ST`**, **`STS`**, **`DF`**, **`DF3`**, **`DFS`**, **`DF3S`**, **`STU or STUS`**).

     – ***`<AddressReference>`*** is the initial address (reference) of the configured type.

     The **Header** field uses the following syntax for using the command Read File Record on the driver sheets:

     `FILE:<ModbusFileNumber>:<InitialRecord>:<RecordSize>:<NumberOfRecords>:<PCDumpFileName>`

     Where:

     - ***`<ModbusFileNumber>`*** is the Modbus File Number which is to be read (1 to 65535)

     - ***`<InitialRecord`***> is the initial record **`within the Modbus File to be read (0 to 10000)`**

     - ***`<RecordSize`***> is the size of a record, in WORDs and is configured on the PLC device

- ***<NumberOfRecords>*** number of records to be read within the Modbus File (1 to 10000). Each record is a word. The numeric value of <InitialRecord> + <NumberOfRecords> cannot exceed 10000.
- ***<PCDumpFileName>*** Name of the CSV file created on the local PC, where the Modbus file records will be saved into. The file will be saved into the \Web sub-folder of the application unless an absolute path is specified. This file will have all the records returned by the read request written as one record per row. The bytes in a record (each row) are separated by comma(,).

  Example — **`FILE:10:20:2:4:DumpFileName.csv`**

After you edit the **Header** field, Studio checks the syntax to determine if it is valid. If the syntax is invalid, then Studio automatically inserts a default value of **`0X:0`**.

You can also specify an indirect tag (e.g. **`{header}`**), but the tag that is referenced must follow the same syntax and contain a valid value.

The following table lists all of the data types and address ranges that are valid for the **Header** field:

Data Types	Sample Syntax	Valid Range of Initial Addresses	Comments
0X	**0X:0**	Varies according to equipment	Coil status: Reads and writes events using Modbus instructions 01, 05, and 15.
1X	**1X:0**	Varies according to equipment	Input status: Reads events using Modbus instruction 02.
STA	**STA:0**	Varies according to equipment	Exception Status: Reads events using Modbus instruction 07.
3X	**3X:0**	Varies according to equipment	Input register: Reads events using Modbus instruction 04.
4X	**4X:0**	Varies according to equipment	Holding register: Reads and writes events using Modbus instructions 03, 06 and 16.
FP	**FP:0**	Varies according to equipment	Floating-point value (Holding Register): Reads and writes floating-point values using two consecutive Holding Registers.
FPS	**FPS:0**	Varies according to equipment	Floating-point value (Holding Register): Reads and writes floating-point values using two consecutive Holding Registers with Byte Swap.
FP3	**FP3:0**	Varies according to equipment	Floating-point value (Input Register): Reads floating-point values using two consecutive Input Registers.
FP3S	**FP3S:0**	Varies according to equipment	Floating-point value (Input Register): Reads floating-point values using two consecutive Input Registers with Byte Swap.
DW	**DW:0**	Varies according to equipment	Dword value (Holding Register): Reads and writes Dword values using two consecutive Holding Registers.
DWS	**DWS:0**	Varies according to equipment	Dword value (Holding Register): Reads and writes Dword values using two consecutive Holding Registers with Byte Swap.
DW3	**DW3:0**	Varies according to equipment	Dword value (Input Register): Reads Dword values using two consecutive Input Registers.
DW3S	**DW3S:0**	Varies according to equipment	Dword value (Input Register): Reads Dword values using two consecutive Input Registers with Byte Swap.
BCD3	**BCD3:0**	Varies according to equipment	BCD value (Input Register): Reads events using Modbus instruction 04.
BCD	**BCD:0**	Varies according to equipment	BCD value (Holding Register): Reads and writes events using Modbus instructions 03, 06 and 16.
BCDDW	**BCDDW:0**	Varies according to equipment	BCD 32-bit integer value (Holding Register): Reads and writes 32-bit integer values using two consecutive Holding Registers.
BCDDWS	**BCDDWS:0**	Varies according to equipment	BCD 32-bit integer value (Holding Register): Reads and writes 32-bit integer values using two consecutive Holding Registers with Byte Swap.
BCDDW3	**BCDDW3:0**	Varies according to equipment	BCD 32-bit integer value (Input Register): Reads 32-bit integer values using two consecutive Input Registers.
BCDDW3S	**BCDDW3S:0**	Varies according to equipment	BCD 32-bit integer value (Input Register): Reads 32-bit integer values using two consecutive Input Registers with Byte Swap.
ID	**ID:0**	Varies according to equipment	Report Slave ID using Modbus instruction 17.
ST	**ST:0**	Varies according to equipment	String value (Holding Register): Reads and writes String values using consecutive Holding Registers.
STS	**STS:0**	Varies according to equipment	String value (Holding Register): Reads and writes String values using consecutive Holding Registers with Byte Swap.

Data Types	Sample Syntax	Valid Range of Initial Addresses	Comments
DF	**`DF:0`**	Varies according to equipment	Double Precision Floating-point Value (Holding Register): Read and Write double precision float-point values using four consecutive Holding Registers.
DF3	**`DF3:0`**	Varies according to equipment	Double Precision Floating-point Value (Input Register): Read double precision float-point values using four consecutive Input Registers.
DFS	**`DFS:0`**	Varies according to equipment	Double Precision Floating-point value (Holding Register): Read and Write double precision floating-point values using four consecutive Holding Registers with Byte Swap.
DF3S	**`DF3S:0`**	Varies according to equipment	Double Precision Floating-point value (Input Register): Read double precision float-point values using four consecutive Input Registers with Byte Swap.
STU	**`STU:0`**	Varies according to equipment	Unicode Strings (Holding Registers): Reads and writes UNICODE strings for consecutive holding registers.
STUS	**`STUS:0`**	Varies according to equipment	Unicode Strings with byte swap (Holding Registers): Reads and writes UNICODE strings with bytes swap within registers for consecutive Holding Registers.

Note:
When you configure DFS or DF3S, the bytes of four registers are swapped.
Example:
7495726.566209 will be read (or written) from (or to) registers as follows. (represented as hex numbers)

DF or DF3	- ACC4 3CA4 0B98 5C41 (Word Swap Off) (Big-Endian)
	- 415C 980B A43C C4AC (Word Swap On) (Little-Endian)
DFS or DF3S	- C4AC A43C 980B 415C (Word Swap Off)
	- 5C41 0B98 3CA4 ACC4 (Word Swap On)

4. For each table row (i.e., each tag/register association), configure the **Address** field using the following syntax…

   For all register types other than **`ST/STS`** (String), use the following syntax:

   `[Signed/Unsigned]<AddressOffset>.[Bit]`

   Examples — **`10`**, **`S20`**, **`U40`**, **`10.5`**

   For **`ST/STS`** (Strings), **`STU/STUS`** (Unicode Strings) registers *only*, use the following syntax:

   `<AddressOffset>:<Length>`

   Example — **`10:5`**

   For the Read File Record command, the address fields are not used. However, it is required to create one integer tag on the driver sheet. This will display the record count in the database spy.

Where:

– ***[Signed/Unsigned]*** (optional): Parameter used for integer values only. Valid values are **S** (Signed) and **U** (Unsigned). If you do not specify this parameter, then Studio uses the default parameter in the *Communication Settings* dialog.

– ***<AddressOffset>***: Parameter that is added to the <***AddressReference>*** parameter of the **Header**, to compose the specific address of the register in the block. The sum of the two parameters cannot equal zero (0); Modbus operands must start in an address that is greater than zero.

– ***[Bit]*** (optional): Use this parameter only for **3X** (Input Register) and **4X** (Holding Register) types, to indicate which bit on the register will be read from and/or written to.

– ***<Length>***: Length of the string (in bytes) to be read or written.

➲ **Attention:**

- The Floating-point value is stored in two consecutive Holding Registers, where the address value corresponds to the first Holding Register position. You must ensure that you do not configure a non-existent address, or a conflict will occur.
- The Floating-point values are 4 bytes using 6 significant digits.
- The Double Floating-point values (DF, DF3) are 8 bytes using 16 significant digits.
- You must not configure a range of addresses greater than the maximum block size (data buffer length) supported by the target device. The default block size is 64 bytes, but this can be changed in the communication settings for the driver.
- For DWord registers (DW, DWS, DW3 and DW3S) using the unsigned option in the Address field, the associated database tag must be Real Type.

For examples of how device registers are specified using **Header** and **Address**, see the following table:

Device Register	Header	Address
00001	`0x:1`	`0`
00010	`0x:0`	`10`
01020	`0x:1000`	`20`
10001	`1x:1`	`0`
10010	`1x:0`	`10`
11020	`1x:1000`	`20`
30001	`3x:1`	`0`
30010	`3x:0`	`10`
31020	`3x:1000`	`20`
30000 and 30001	`FP3:0`	`0`
31001 and 31000	`FP3S:0`	`1000`
40001	`4x:1`	`0`
40010	`4x:0`	`10`
41020	`4x:1000`	`20`
40010 (bit 0)	`4x:0`	`10.0`
41010 (bit 7)	`4x:1000`	`10.7`
40001 and 40002	`FP:1`	`0`
40013 and 40014	`FP:0`	`13`
41021 and 41022	`FP:1000`	`21`
40001 and 40002	`DW:1`	`0`
40013 and 40014	`DW:0`	`13`
40001 to 40004	`DF:0`	`1`
30001 to 30004	`DF3:1`	`0`
40001 to 40004	`DFS:0`	`1`
30001 to 30004	`DF3S:1`	`0`
40010	`STU:0`	`10.10`
40100	`STUS:0`	`100.6`

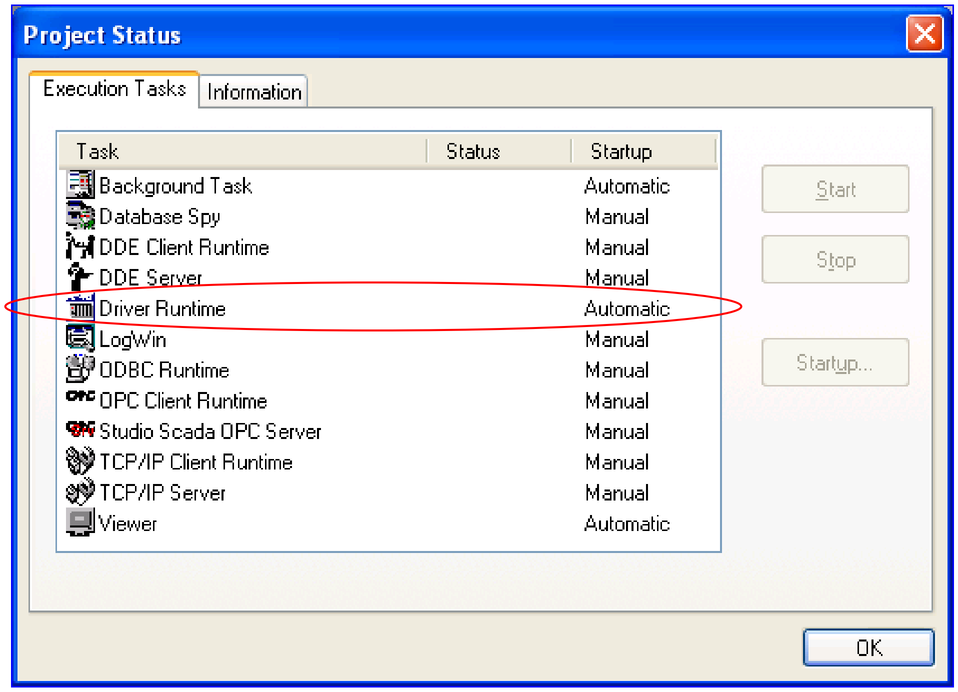
For more information about the device registers and addressing, please consult the manufacturer's documentation.

Executing the Driver

By default, Studio will automatically execute your selected communication driver(s) during application runtime. However, you may verify your application's runtime execution settings by checking the *Project Status* dialog.

To verify that the communication driver(s) will execute correctly:

1. From the main menu bar, select **Project** → **Status**. The *Project Status* dialog displays:



Task	Status	Startup
Background Task		Automatic
Database Spy		Manual
DDE Client Runtime		Manual
DDE Server		Manual
Driver Runtime		Automatic
LogWin		Manual
ODBC Runtime		Manual
OPC Client Runtime		Manual
Studio Scada OPC Server		Manual
TCP/IP Client Runtime		Manual
TCP/IP Server		Manual
Viewer		Automatic

Project Status Dialog

2. Verify that the *Driver Runtime* task is set to **Automatic**.
   - If the setting is correct, then proceed to step 3 below.
   - If the **Driver Runtime** task is set to **Manual**, then select the task and click the **Startup** button to toggle the task's *Startup* mode to **Automatic**.
3. Click **OK** to close the *Project Status* dialog.
4. Start the application to run the driver.

Troubleshooting

If the MODBU driver fails to communicate with the target device, then the database tag(s) that you configured for the **Read Status** or **Write Status** fields of the Main Driver Sheet and Standard Driver Sheet will receive an error code. Use this error code and the following table to identify what kind of failure occurred.

Error Code	Description	Possible Causes	Procedure to Solve
0	OK	Communication without problems	None required
1	Invalid Request	Action not allowed for slave.	▪ Check Slave version, this function is only applicable to newer devices and was not implemented in the unit selected. ▪ Check slave configuration and if state is OK to process request. ▪ If a Poll Program Complete command was issued this code indicates that no program function preceded it.
2	Illegal Data Value	Try to read an address that is not present in the PLC, such as 49999	Type a valid address.
3	Invalid data values	Specified address does not exist on the device so protocol received invalid data	Verify that specified address exists on the device.
4	Equipment failed	Equipment failed or out of order	Check equipment state.
5	Ack	The PLC may be busy and sent this Ack to acknowledge that it received the message but is unable to respond at the time	Wait until the PLC is available again and restart communicating with it
6	Equipment in use	Command invalid when equipment is in use	Studio commands cannot generate this error.
7	Negative Ack	Ack action error during communication	Check device and Studio Communication Parameters.
8	Memory parity error	Invalid Communication Parameter	Check driver Communication Parameters.
10	Invalid Header field	Specified invalid tag value in Header field	Specify a valid Header tag value.
11	Invalid Address field	Specified invalid Address	Specify a valid address.
12	Invalid block size	Offset greater than maximum allowed	Specify a valid offset or create a new worksheet. Typically, maximum offset is 64.
13	Invalid CRC	Invalid CRC in response message	▪ Check the cable wiring. ▪ Check the station number. ▪ Check the RTS/CTS configuration (see Studio *Technical Reference Manual* for valid configurations).
18	Invalid BCD Value	Tried reading an invalid BCD value	Verify that PLC value is valid.
19	Invalid BCD Value	Tried writing a negative BCD value	Only positive BCD values are valid.
100	Illegal Operation	Tried to write to Read Only addresses (1x and 3x)	Writing operations are possible only in Coil Status and Holding Registers areas
-15	Timeout Start Message	▪ Disconnected cables ▪ PLC is turned off, in stop mode, or in error mode ▪ Wrong station number ▪ Wrong RTS/CTS control settings	▪ Check cable wiring. ▪ Check the PLC state – it must be RUN. ▪ Check the station number. ▪ Check the RTS/CTS configuration (see Studio *Technical Reference Manual* for valid configurations). ▪ Check the maximum registers configuration (see Studio *Technical Reference Manual* for valid configurations).
-17	Timeout between rx char	▪ PLC in stop mode or in error mode ▪ Wrong station number ▪ Wrong parity	▪ Check cable wiring. ▪ Check the PLC state – it must be RUN. ▪ Check the station number.

		▪ Wrong RTS/CTS configuration settings	▪ Check the RTS/CTS configuration (see "Network Specifications" for valid RTS/CTS configurations). ▪ Check the maximum registers configuration (see Studio *Technical Reference Manual* for valid configurations).
20	Read out of sync	▪ Timeout value	▪ Increase timeout value
21	Error in retrieving all File records requested using Read File Record command.	▪ This is an error with the local dumpfile either in creating the file or writing to the file.	▪ Check the file path given and try again.
22	File records are not supported in MDS and writing on Driver sheets	▪ Attempted to perform the Read File Record command on MDS or a Write File Record on MDS / Driver Sheet	▪ Read File Record is supported only on the Driver sheets. Write File Record is not supported at all even on the driver sheets.

⇨ **Tip:**

You can monitor communication status by establishing an event log in Studio's *Output* window (*LogWin* module). To establish a log for **Field Read Commands**, **Field Write Commands** and **Serial Communication,** right-click in the *Output* window and select the desired options from the pop-up menu.

You can also use the *LogWin* module (**Tools** → **LogWin**) to establish an event log on a remote unit that runs Windows CE. The log is saved on the unit in the `celog.txt` file, which can be downloaded later.

If you are unable to establish communication between Studio and the target device, then try instead to establish communication using the device's own programming software (e.g., ModSoft). Quite often, communication is interrupted by a hardware or cable problem or by a device configuration error. If you can successfully communicate using the programming software, then recheck the driver's communication settings in Studio.

If you must contact us for technical support, please have the following information available:

- **Operating System** (type and version): To find this information, select **Tools** → **System Information**.
- **Project Information**: To find this information, select **Project** → **Status.**
- **Driver Version** and **Communication Log**: Displays in the Studio *Output* window when the driver is running.
- **Device Model** and **Boards**: Consult the hardware manufacturer's documentation for this information.

Sample Application

A sample application that employs the MODBU driver is provided on the Studio installation CD. We strongly recommend that you use this sample application to test the driver *before* you develop your own applications, for the following reasons:

- To better understand the information and instructions provided in this document;
- To verify that your driver configuration is working satisfactorily with the target device; and
- To ensure that the all of hardware used in the test (i.e. the device, adapter, cable, and PC) is functioning safely and correctly.

Note:
The following instructions assume that you are familiar with developing project applications in Studio. If you are not, then please review the relevant chapters of the Studio *Technical Reference Manual* before proceeding.

To use the sample application:

1. Configure the device's communication settings according to the manufacturer's documentation.
2. Run Studio.
3. From the main menu bar, select **File → Open Project**.
4. Insert the Studio installation CD and browse it to find the sample application. It should be located in the directory **`\COMMUNICATION EXAMPLES\MODBU`**.
5. Select and open the sample application.
6. Configure and test the driver, as described in the rest of this document.

When you have thoroughly tested the driver with your target device, you may proceed with developing your own Studio application projects.

Tip:
You can use the sample application screen as the maintenance screen for your own applications.

Revision History

Doc. Revision	Driver Version	Author	Date	Description of Changes
A	2.02	Roberto V. Junior	Jul/30/1999	▪ First driver version ▪ Driver available for Windows CE
B	2.03	Roberto V. Junior	Dec/13/1999	Added Read and Write of Floating-point operand
C	2.04	Roberto V. Junior	Jun/5/2000	Added CRC verification of device response
D	2.05	Lourenço Teodoro	Oct/30/2000	Added MAIN DRIVER SHEET feature
E	2.06	Lourenço Teodoro	Nov/14/2001	▪ Added FPS, FP3, and FP3S data type ▪ Changed Ero functionality
F	2.06	Fabíola Fantinato	Dec/3/2001	Revised document to conform to documentation standards
G	2.07	Roberto V. Junior	Feb/21/2002	▪ Added bit read/write to 4x and 3x data type ▪ Fixed bug that caused 100% CPU usage when a message was waiting in a timeout situation ▪ Added a warning message in the LogWin when the AddressReference in Header field plus the lowest AddressOffset in Address column is equal to zero
H	2.08	Eric Vigiani	Aug/22/2003	▪ Added DW data type
I	2.09	Eric Vigiani	Dec/11/2003	▪ Implemented Signed/Unsigned option by address ▪ Added DW3 data type
J	2.10	Eric Vigiani	May/25/2004	Implemented writing group commands when writing FP values
K	2.11	Fábio H.Y. Komura	Jun/14/2004	▪ Added DWS and DW3S data types ▪ Implemented SwapWord for FP, FPS, FP3, FP3S, DW, DWS, DW3 and DW3S ▪ Changed FP, FPS, FP3, FP3S, DW, DWS, DW3 and DW3S with and without SwapWord to conformance to standards (FPS, FP3S, DWS and DW3S are data types with Byte Swap)
L	2.12	Fábio H.Y. Komura	Sep/03/2004	Added BCD, BCD3, BCDDW, BCDDWS, BCDDW3 and BCDDW3S data type.
M	2.13	Fábio H.Y. Komura	Sep/24/2004	Fixed problem with Parser Address
N	2.14	Leandro Coeli	Jan/19/2005	Insert String type
O	2.15	Leandro Coeli	Feb/19/2005	Implemented LRC to ASCII communication
P	2.16	Leandro Coeli	Apr/26/2005	Implemented configurable Block Size
Q	2.18	Graziane C. Forti	Jun/12/2006	▪ Implemented to check the Station into the Rx messages. ▪ Implemented the Unsigned in all header ▪ Implemented Block size 1 using MDS
R	2.18	Michael D. Hayden	Aug/22/2006	Edited for language and usability.
S	2.19	Eric Vigiani	Aug/31/2006	Fixed problem with the ST header.

Doc. Revision	Driver Version	Author	Date	Description of Changes
T	2.20	Graziane C. Forti	Nov/28/2006	▪ Implemented STS Header ▪ Fixed problem BCDDW, BCDDWS, BCDDW3, BCDW3S and ST using Block Size=1 ▪ Fixed problem S/U address configuration (types that this operation is not allowed) ▪ Implemented Block size 1 using MDS ▪ Fixed problem ST/STS using write trigger (SDS) ▪ Fixed problem to calculate LRC – ASCII protocol ▪ Changed Maximum Range (exception problem) ▪ Fixed the String writing
U	2.21	Graziane C. Forti	Jan/19/2007	▪ Implemented Unsigned/Signed to work like version 2.16 as well, using ComboBox
V	2.22	Eric Vigiani	Feb/21/2008	▪ Removed the error Invalid Word Swap for ST and STS header.
W	2.22	Andre Bastos	Dec/29/2008	▪ Removed Cable Wiring Scheme from doc
X	10.1	Marcelo Carvalho	Jan/07/2009	▪ Updated driver version, no changes in the contents.
Z	10.3	Fellipe Peternella	Jul/1/2009	▪ Modified driver to properly handle error codes sent by the PLC ▪ Modified driver to support communication with multiple Stations when using Serial Encapsulation over TCP/IP or UDP/IP
AA	10.4	André Körbes	Sep/16/2010	▪ Improved driver reliability.
AB	10.5	Paulo Balbino	Jan/17/2013	▪ Fixed problem of out of sync after timeout. Writing incorrect values to tags.
AC	10.6	Paulo Balbino	Sept/23/3012	▪ Fixed issue reading coils ▪ Fixed issue of requesting more words than necessary when communicating with ST and STS datatypes
AD	10.7	Caio Cerquetani	Dec/11/2013	▪ Fixed bug of data writing
AE	10.8	Charan Manjunath	Mar/05/2014	▪ Fixed issue of getting the error code on write operations.
AF	10.9	Felipe Andrade	Oct/17/2014	▪ Added support for headers DF, DF3, DFS and DF3S.
AG	10.10	Priya Yennam	Jan/30/2015	▪ Added support for headers STU, STUS for UNICODE strings
AH	10.11	Priya Yennam Anushree Phanse	Sep/19/2016	▪ Implemented support for Read File Record on Driver Sheets ▪ Ported driver to be platform agnostic.
AI	10.12	Anushree Phanse	Apr/03/2017	▪ Fixed issue of Modbu Serial Driver not communicating properly on Raspberry Pi. ▪ Improved documentation to provide more information about how to use the driver on Raspberry Pi, how to find and configure the serial and USB ports on the linux devices and how to create and use multiple instances of the driver on windows and linux devices.
AJ	10.12	Anushree Phanse	Jun/08/2017	▪ Improved driver documentation to include accurate information about using indirect tags on MDS. No change in the driver

Appendix A – Using MODBU on Raspberry Pi 3

The MODBU Serial driver can be run on Linux devices using *IotView*. See *User Guide and Technical Reference for Wonderware Indusoft Web Studio* on how to configure and install *IotView* on different Linux devices.

This section describes how to configure the port on Raspberry Pi 3 to be able to successfully communicate using MODBU.

1. Run *RemoteAgent* on the Raspberry Pi and connect to it using *Remote Management*.
   The *Platform* should be ***arm-gnueabihf-2.13-6.0.17***. Click on *Install System Files*.

2. On the Raspberry Pi create a document with the name **99-port.rules**. When in the location of the folder from which the RemoteAgent is running on the terminal follow these steps to create and populate this file.

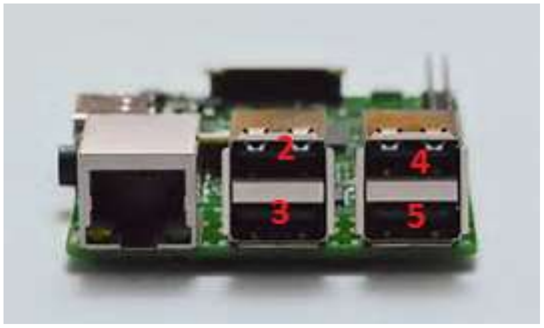
```
$ sudo cat > 99-port.rules
 #Link the Raspberry Pi ports to the short name.
SUBSYSTEMS=="usb", ENV{ID_PATH}=="platform-3f980000.usb-usb-0:1.2:1.0", SYMLINK+="piusb2"
SUBSYSTEMS=="usb", ENV{ID_PATH}=="platform-3f980000.usb-usb-0:1.3:1.0", SYMLINK+="piusb3"
SUBSYSTEMS=="usb", ENV{ID_PATH}=="platform-3f980000.usb-usb-0:1.4:1.0", SYMLINK+="piusb4"
SUBSYSTEMS=="usb", ENV{ID_PATH}=="platform-3f980000.usb-usb-0:1.5:1.0", SYMLINK+="piusb5"
```

Then press Ctrl+D to save the file

3. Then use the following command to copy the newly created file 99-port.rules to the correct location

```
$ sudo cp 99-port.rules /lib/udev/rules.d/
```

4. Reboot Raspberry Pi. The file 99-port.rules will configure the following persistent USB port mapping:



Based on which USB port is being used (using USB to Serial convertors to talk to MODBUS devices) use the following convention to configure the Serial Port setting on the driver settings:

USB Port 2:	/dev/piusb2
USB Port 3:	/dev/piusb3
USB Port 4:	/dev/piusb4
USB Port 5:	/dev/piusb5

Refer to the section ***Configuring the Communication Settings*** in this document for more information.

4. Start the *RemoteAgent* on the Raspberry Pi and using *Remote Management* on Wonderware Indusoft Web Studio perform a *Download* of the project. Start *IoTView Runtime*, the driver should be able to successfully communicate with the Modbus device connected to the appropriate USB port of the Rapsberry Pi device.

Appendix B – Find and configure the serial ports that can be used on Linux devices

1. Connect Serial cable or USB to Serial Convertor cable to the Linux device like Ubuntu or Raspberry Pi respectively and then to the Modbus device.

2. Run the following in the command line:

```
$ dmesg | grep –i ttys
```

3. It will display the serial ports available on the device like:

ttyS1, ttyS2 etc. on an Ubuntu device with serial ports
or
ttyUSB0, ttyUSB1 etc. on a Raspberry Pi.

If the Step 1 is correctly followed the response to the following command, should also denote which of ports the cable is connected to.

4. If using a port ***ttyS1***, use the syntax ***/dev/ttyS1*** on the COM port setting on the driver setting in the development environment

If using a port ***ttyUSB0***, use the syntax ***/dev/ttyUSB0*** on the COM port setting on the driver setting in the development environment.

Note:
In case of the Raspberry Pi if you have followed instructions from *Appendix A: Using MODBU on Raspberry Pi 3* You can skip this step, and use the persistent ports as shown in the picture by configuring ***/dev/piusb2*** etc.

Appendix C – Running multiple instances of MODBU to communicate with devices connected to multiple USB/Serial ports

This section will describe how to create multiple MODBU drivers on the same Windows/Linux device to communicate with MODBUS devices connected to its multiple USB/Serial ports.

To run multiple instances of MODBU driver on Windows devices:

1. Find the product installation path for Wonderware Indusoft Webstudio, using the function *GetProductPath()* on Database Spy. Close Wonderware Indusoft Web Studio. On Windows Explorer browse to the product installation path. Open the \DRV subdirectory inside this path.

2. Duplicate the files *MODBU.dll, MODBU.ini, MODBU.msg* and *MODBU.pdf* for as many instances of the driver required to be created. Rename the duplicated files *like MODB1.dll, MODB1.ini, MODB1.msg and MODB1.pdf* etc.

3. Open the Wonderware Indusoft Web Studio. See section ***Selecting the Driver*** in this document to browse all the drivers available. The *Available Drivers List* should show the new instances of MODBU driver created like MODBU1 etc. See section ***Configuring the Communication Settings*** in this document to successfully add and configure the new instances of MODBU driver like MODB1 etc.

4. New instances of MODBU driver like MODBU1 should successfully start communication when the project Runtime is started. If running the drivers on a WinCE or WinEmbedded devices: perform an *Install System files* again on the target devices using *Remote Management*. For more information about using WinCE and WinEmbedded devices refer to *User Guide and Technical Reference for Wonderware Indusoft Web Studio*.

To run multiple instances of MODBU driver on Linux devices:

1. Repeat Steps 1- 3 described above (*To run multiple instances of MODBU driver on Windows devices*).

If using a Raspberry Pi 3 with USB to Serial connections, repeat steps 1-3 described in section ***Appendix A: Using MODBU on Raspberry Pi 3.***

See *User Guide and Technical Reference for Wonderware Indusoft Web Studio* on how to configure and install *IotView* on you Linux devices and follow instructions to install *IotView* on the specific platform.

2. Duplicate the MODBU driver on the Linux device using the following commands from the folder where the *RemoteAgent* is located on the Linux device. Duplicate as many instances as required.

```
$ sudo cp ./drv/modbu.so ./drv/modb1.so
$ sudo cp ./drv/modbu.ini ./drv/modb1.ini
$ sudo cp ./drv/modbu.msg ./drv/modb1.msg
```

3. On Wonderware Indusoft Web Studio Development check that each of the duplicated instances of the MODBU drivers has the correct Serial Port configuration as used by the Linux device.

4. Perform a *Download* of the project to the Linux device using *Remote Management* and start the *IoTView Runtime*. The multiple instances of the MODBU driver should communicate successfully with the MODBUS devices connected to multiple serial ports or USB to Serial ports on the Linux device.