MOTCP Communication Driver

Driver for TCP/IP Communication
with Devices Using Modbus Protocol

Contents

Introduction

The MOTCP driver enables communication between the Studio system and devices using the Modbus protocol over TCP/IP, according to the specifications discussed in this document.

This document will help you to select, configure and execute the MOTCP 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 MOTCP driver in the Studio system.
- **Configuring the Device**: Describes how the target device must be configured to receive communication from the MOTCP driver.
- **Configuring the Driver**: Explains how to configure the MOTCP driver in the Studio system, including how to associate database tags with device registers.
- **Executing the Driver**: Explains how to execute the MOTCP driver during application runtime.
- **Troubleshooting**: Lists the most common errors for this driver, their probable causes, and basic procedures to resolve them.
- **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 environment.

General Information

This chapter identifies all of the hardware and software components required to implement communication between the MOTCP driver in Studio and a target device using the Modbus protocol over TCP/IP.

The information is organized into the following sections:

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

Device Specifications

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

- **Manufacturer**: Modicon or any device using the Modbus protocol communicating over TCP/IP
- **Compatible Equipment**:
  - Any device that is fully compatible with the Modbus protocol
- **Device Runtime Software**: None

For a description of the device(s) used to test driver conformance, see “Conformance Testing” on the next page.

Network Specifications

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

- **Device Communication Port**: Modbus Ethernet Port
- **Physical Protocol**: Ethernet
- **Logic Protocol**: Modbus over TCP/IP
- **Specific PC Board**: Any TCP/IP Adapter (Ethernet board)

Driver Characteristics

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

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

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

- Windows XP/7/8/2003/2008/2012
- Windows CE
- Windows Embedded

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

The MOTCP 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)	1 Word	–	●	–	–	–	–	–	–	–

Conformance Testing

The following hardware/software was used for conformance testing:

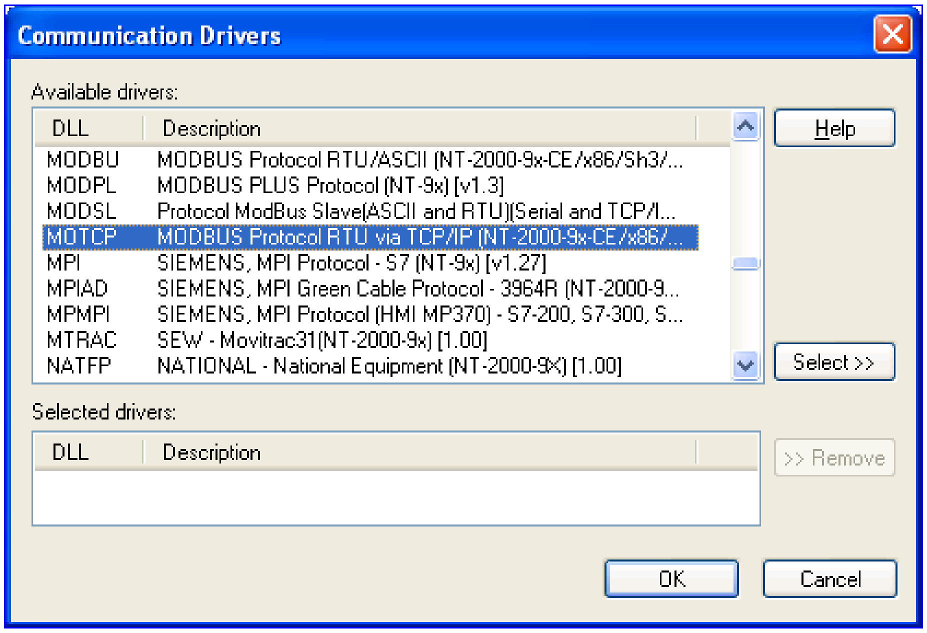
- **Driver Configuration**:
  - **Protocol**: RTU
  - **Cable**: Ethernet Cable

Driver Version	Studio Version	Operating System (development)	Operating System (runtime)	Equipment
10.19	8.1 + SP2	Windows 7/8	▪ Win8 x64 ▪ WinCEv7.00	Wago 750-841 CPU GE-FANUC 90-30 CPU-374 Schneider/Modicon TSX Quantum with NOE 211 00 module Modbus Simulator

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 MOTCP driver for your Studio application:

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



DLL	Description
MODBU	MODBUS Protocol RTU/ASCII (NT-2000-9x-CE/x86/Sh3/...
MODPL	MODBUS PLUS Protocol (NT-9x) [v1.3]
MODSL	Protocol ModBus Slave(ASCII and RTU)(Serial and TCP/I...
MOTCP	MODBUS Protocol RTU via TCP/IP (NT-2000-9x-CE/x86/...
MPI	SIEMENS, MPI Protocol - S7 (NT-9x) [v1.27]
MPIAD	SIEMENS, MPI Green Cable Protocol - 3964R (NT-2000-9...
MPMPI	SIEMENS, MPI Protocol (HMI MP370) - S7-200, S7-300, S...
MTRAC	SEW - Movitrac31(NT-2000-9x) [1.00]
NATFP	NATIONAL - National Equipment (NT-2000-9X) [1.00]

Communication Drivers Dialog

3. When the **MOTCP** 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.

Note:
It is not necessary to install any other software on your computer to enable communication between Studio and your target device. However, this communication can only be used by the Studio application; it cannot be used to download control logic to the device. To download control logic to a Modbus device, you must also install the Modbus programming software (e.g., ModSoft). For more information, please consult the documentation provided by the device manufacturer.

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 many different types of devices that use the Modbus protocol, we cannot define a standard device configuration. Consequently, we recommend using the device manufacturer's suggested configuration. Please consult the manufacturer's documentation.

Once the selected driver and the target device are both properly configured, it is not necessary to install any other software on your computer to enable communication between the host and the device. All runtime communication is handled within your Studio application project. However, programming the device itself — that is, developing control logic and downloading it to the device — still requires using the device's own programming tool.

Configuring the Driver

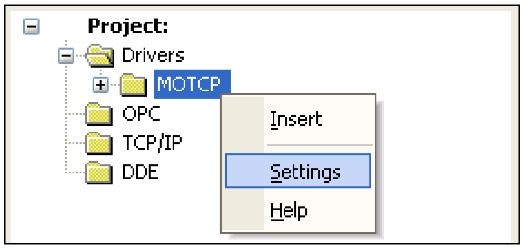
Once you have selected the MOTCP driver in Studio, you must properly configure it to communicate with your target device. First, you must set the driver's communication settings to match the parameters set on the device. Then, you must build driver worksheets to associate database tags in your Studio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

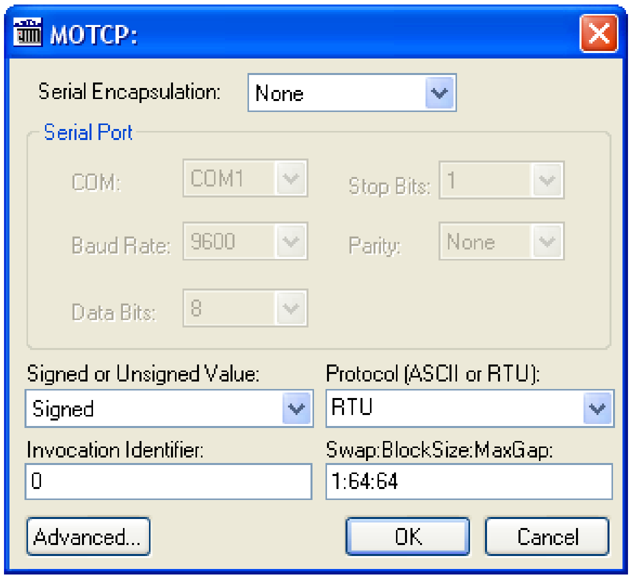
For the purposes of this document, only MOTCP driver-specific settings and procedures will be discussed here. To configure the communication settings for the MOTCP driver:

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



Select Settings from the Pop-Up Menu

The *MOTCP: Communication Settings* dialog is displayed:



MOTCP:

Serial Encapsulation: None

Serial Port

COM: COM1 | Stop Bits: 1

Baud Rate: 9600 | Parity: None

Data Bits: 8

Signed or Unsigned Value: Signed

Protocol (ASCII or RTU): RTU

Invocation Identifier: 0

Swap:BlockSize:MaxGap: 1:64:64

MOTCP: Communication Settings Dialog

3. In the *Communication Settings* dialog, configure the driver settings to enable communication with your target device. To ensure error-free communication, the driver settings must *exactly match* the corresponding settings on the device. Please consult the manufacturer's documentation for instructions how to configure the device and for complete descriptions of the settings.

   Depending on your circumstances, you may need to configure the driver *before* you have configured your target device. If this is the case, then take note of the driver settings and have them ready when you later configure the device.

   ➲ **Attention:**
   For safety reasons, you **must** take special precautions when connecting and configuring new equipment. Please consult the manufacturer's documentation for specific instructions.

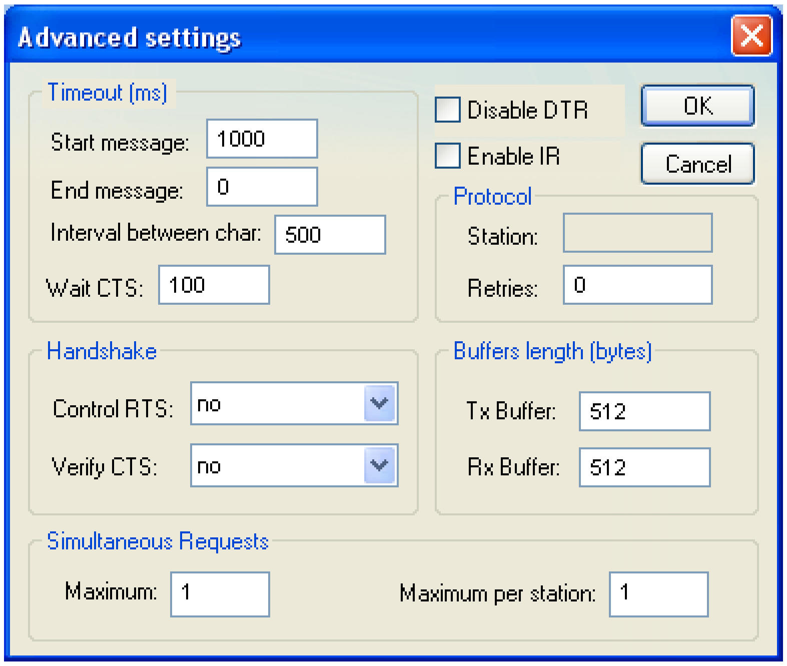
The communication settings and their possible values are described in the following table:

Parameters	Default Values	Valid Values	Description
Signed or Unsigned Value	`Signed`	**Signed or Unsigned**	How integer values are handled by the device: ▪ **Signed** – Integers can be positive or negative ▪ **Unsigned** – Integers are neither positive nor negative
Invocation Identifier	`0`	`-1` to `255`	Used for transaction pairing when multiple Words are sent along the same connection without waiting for a response. When specifying a **negative number**, the invocation identifier byte will be automatically incremented otherwise the specified value will be the one sent as the invocation identifier.
Protocol (ASCII or RTU)	**`RTU`**	**`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.
		`ASCII`	Each eight-bit Word is sent as two four-bit ASCII characters, allowing for a time interval between characters without causing errors.
Swap:Block Size:MaxGap	`0:64:10`	`0` or `1` or `2` (Swap)	Option to change the order of data as they are processed: ▪ **0** – Word Swap OFF; registers are not swapped. ▪ **1** – Word Swap ON; swap Words for **FP**, **FPS**, **FP3**, **FP3S**, **DW**, **DWS**, **DW3**, **DW3S**, **BCDDW**, **BCDDWS**, **BCDDW3**, **BCDDW3S**, **DF**, **DF3**, **DFS** and **DF3S** register types. ▪ **2** – Legacy Swap; swap Words for **FPS**, **FP3S**, **DWS**, **DW3S**, **BCDDWS**, **BCDDW3S, DFS** and **DF3S** register types only. The other register types are not swapped See note below.
			Note: - The parameter 1 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) - The parameter 2 just will affect **DFS** and **DF3S** in a different way, converting their bytes order from Big-Endian to Little-Endian. (example in page 15)
		`0` to `512` (Block Size)	The nominal size (in Words) of each block of data to be transmitted, as determined by the processing capacity of the device. Usually, Modicon devices support up to 125 words.
		`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**.

➲ **Attention**:
When you configure Legacy Swap in the **Swap** field of *Communication Settings*, the registers FPS, FP3S, DWS, DW3S, BCDDW and BCDDW3S do Word Swap (not Byte Swap). This is in compliance with the older versions of MODBU driver (v2.10 or older). The others registers are not swapped.

Note:
The device must be configured with *exactly the same* parameters that you configured in the *MOTCP Communication Parameters* dialog.

4. In the *Communication Settings* dialog, click the **Advanced** button to open the *Advanced Settings* dialog:


Advanced settings

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

Simultaneous Requests
Maximum: 1
Maximum per station: 1

Advanced Settings Dialog

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.

5. 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.

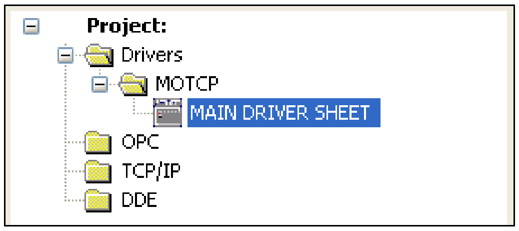
The configuration of these worksheets is described in detail in the “Communication” chapter of the Studio *Technical Reference Manual*, and the same general procedures are used for all drivers. Please review those procedures before continuing.

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

MAIN DRIVER SHEET

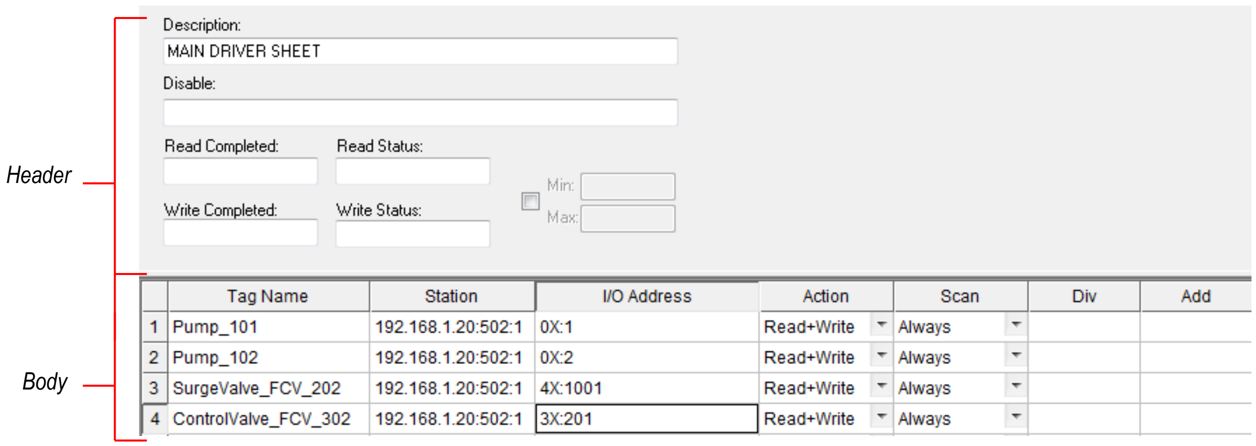
When you select the MOTCP driver and add it to your application, Studio automatically inserts the *Main Driver Sheet* in the *MOTCP* driver subfolder. To configure the Main Driver Sheet:

1. Select the *Comm* tab in the *Workspace* pane.
2. Open the *Drivers* folder, and then open the *MOTCP* subfolder:



Main Driver Sheet in the MOTCP Subfolder

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



Description: MAIN DRIVER SHEET
Disable:
Read Completed:
Read Status:
Write Completed:
Write Status:

	Tag Name	Station	I/O Address	Action	Scan	Div	Add
1	Pump_101	192.168.1.20:502:1	0X:1	Read+Write	Always		
2	Pump_102	192.168.1.20:502:1	0X:2	Read+Write	Always		
3	SurgeValve_FCV_202	192.168.1.20:502:1	4X:1001	Read+Write	Always		
4	ControlValve_FCV_302	192.168.1.20:502:1	3X:201	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 MOTCP 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 IP Address of the device, using the following syntax:

  `<IP Address>:<Port Number>:<PLC ID>`

  Example — **`192.168.125.31:502:1`**

  Where:

  – ***`<IP Address>`*** is the device's IP address on the TCP/IP network;

  – ***`<Port Number>`*** is the port number for the Modbus TCP protocol (usually 502); and

  – ***`<PLC ID>`*** is the PLC identification number (from 1 to 254).

  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 must use a non-zero value in the **Station** field, and you cannot leave the field blank.

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

  For all register types other than **`ST`** and **`STS`** use the following syntax:

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

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

  For **`ST`**, **`STS, STU and STUS`** registers *only*, use the following syntax:

  `<Type>:<Address>:<Length> or`

  `<Type>:<Address>.[StartByte]:<Length>`

  Example — **`ST:10:5`**

  Where:

  – ***<Type>*** Register type. Valid values are (**0X**, **1X**, **3X**, **4X**, **FP, FPSW, FP3, FP3SW, DF, DF3, DFS, DF3S, DF3S, DW**, **FPS**, **DWS**, **DW3**, **DW3S, DWSW**, **BCD**, **BCD3**, **BCDDW**, **BCDDWS**, **BCDDW3**, **BCDDW3S**, **ID**, **ST**, **STS**, **HRW, STU, 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.

  – ***`[StartByte]`*** (optional): Use this parameter only for **`ST`** and **`STS`**, to indicate the initial byte.

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

> ➲ **Attention:**
> - This driver supports bit reading only; it cannot execute bit writing. Also, when an unsigned DWord register type is specified, it can only be associated with a Real database tag in Studio.
> - The Floating-point values are 4 bytes using 6 significant digits.
> - Modbus operands must start in an address that is greater than zero.
> - Main Driver Sheet does not support the Read/Write of File Records.

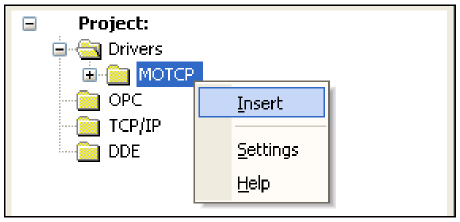
STANDARD DRIVER WORKSHEET

When you select the MOTCP 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.

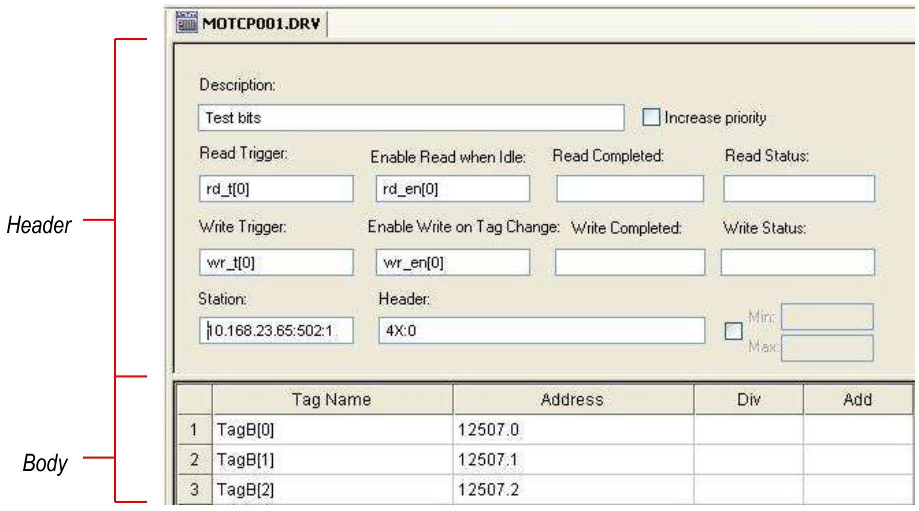
To insert a new Standard Driver Worksheet:

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



Inserting a New Worksheet

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



Description:	Test bits
Increase priority	☐
Read Trigger:	rd_t[0]
Enable Read when Idle:	rd_en[0]
Read Completed:	
Read Status:	
Write Trigger:	wr_t[0]
Enable Write on Tag Change:	wr_en[0]
Write Completed:	
Write Status:	
Station:	10.168.23.65:502:1
Header:	4X:0
Min:	
Max:	

	Tag Name	Address	Div	Add
1	TagB[0]	12507.0		
2	TagB[1]	12507.1		
3	TagB[2]	12507.2		

MOTCP Driver Worksheet

> **Note:**
> Worksheets are numbered in order of creation, so the first worksheet is `MOTCP001.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 MOTCP driver.

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

- **Station** field: Specify the IP Address of the device and the slot number, using the following syntax:

  <IP Address>*:*<Port Number>*:*<PLC ID>

  Example — **192.168.125.31:502:1**

  Where:

  – ***<IP Address>*** is the device’s IP address on the TCP/IP network;

  – ***<Port Number>*** is the port number for the Modbus TCP protocol (usually 502); and

  – ***<PLC ID>*** is the PLC identification number (from 1 to 254).

  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.
> • Modbus operands are base 1, which means that there are no 0 addresses. If you configure the header with an Address Reference as 0, the address offset must be greater than 0.

- **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>*** Register type. Valid values are (**0X**, **1X**, **3X**, **4X**, **FP, FPSW, FP3, FP3SW, DF, DF3, DFS, DF3S, FP3S, DW**, **FPS**, **DWS**, **DW3**, **DW3S, DWSW**, **BCD**, **BCD3**, **BCDDW**, **BCDDWS**, **BCDDW3**, **BCDDW3S**, **ID**, **ST**, **STS, HRW, STU, 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 (0x20)** on the driver sheets:

  FILE:<ModbusFileNumber>:<InitialRecord>:<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)`**
- ***<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:10:20:4: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 a string tag (e.g. **`{header}`**), but the tag value 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 MOTCP driver:

Type	Syntax	Valid Range	Comments
0X	**0X:0**	varies by device	Coil Status: Read and write events using Modbus instructions 01, 05 and 15.
1X	**1X:0**	varies by device	Input Status: Read events using Modbus instruction 02.
3X	**3X:0**	varies by device	Input Register: Read events using Modbus instruction 04.
4X	**4X:0**	varies by device	Holding Register: Read and write events using Modbus instructions 03, 06 and 16.
FP	**FP:0**	varies by device	Floating-point Value (Holding Register): Read and write float-point values using two consecutive Holding Registers.
FPSW	**FPSW:0**	varies by device	Floating-point Value (Holding Register): Read and write float-point values using two consecutive Holding Registers with Word Swap.
FP3	**FP3:0**	varies by device	Floating-point Value (Input Register): Read float-point values using two consecutive Input Registers.
FP3SW	**FP3SW:0**	varies by device	Floating-point Value (Input Register): Read float-point values using two consecutive Holding Registers with Word Swap.
DF	**DF:0**	varies by device	Double Precision Floating-point Value (Holding Register): Read and Write double precision float-point values using four consecutive Holding Registers.
DF3	**DF3:0**	Varies by device	Double Precision Floating-point Value (Input Register): Read double precision float-point values using four consecutive Input Registers.
FP3S	**FP3S:0**	varies by device	Floating-point Value (Input Register): Read float-point values using two consecutive Input Registers with Byte Swap. (*)
DW	**DW:0**	varies by device	32-bit Integer Value (Holding Register): Read and write 32-bit integer values using two consecutive Holding Registers.
FPS	**FPS:0**	varies by device	Floating-point Value (Holding Register): Read and write float-point values using two consecutive Holding Registers with Byte Swap. (*)
DWS	**DWS:0**	varies by device	32-bit Integer Value (Holding Register): Read and write 32-bit integer values using two consecutive Holding Registers with Byte Swap. (*)
DW3	**DW3:0**	varies by device	32-bit Integer Value (Input Register): Read 32-bit integer values using two consecutive Input Registers.
DW3S	**DW3S:0**	varies by device	32-bit Integer Value (Input Register): Read 32-bit integer values using two consecutive Input Registers with Byte Swap. (*)
DWSW	**DWSW:0**	Varies by device	32-bit Integer Value (Holding Register): Read and write 32-bit integer values using two consecutive Holding Registers with Word Swap. (*)
BCD3	**BCD3:0**	varies by device	BCD Value (Input Register): Read events using Modbus instruction 04.
BCD	**BCD:0**	varies by device	BCD Value (Holding Register): Read and write events using Modbus instructions 03, 06 and 16.
BCDDW	**BCDDW:0**	varies by device	BCD 32-bit Integer Value (Holding Register): Read and write 32-bit integer values using two consecutive Holding Registers.
BCDDWS	**BCDDWS:0**	varies by device	BCD 32-bit Integer Value (Holding Register): Read and write 32-bit integer values using two consecutive Holding Registers with Byte Swap. (*)
BCDDW3	**BCDDW3:0**	varies by device	BCD 32-bit Integer Value (Input Register): Read 32-bit integer values using two consecutive Input Registers.

Type	Syntax	Valid Range	Comments
BCDDW3S	**`BCDDW3S:0`**	varies by device	BCD 32-bit Integer Value (Input Register): Read 32-bit integer values using two consecutive Input Registers with Byte Swap. (*)
ID	**`ID:0`**	varies by device	Report Slave ID using Modbus instruction 17.
ST	**`ST:0`**	varies by device	String value (Holding Register): Reads and writes String values using consecutive Holding Registers.
STS	**`STS:0`**	varies by device	String value (Holding Register): Reads and writes String values using consecutive Holding Registers with Byte Swap.
HRW	**`HRW:0`**	varies by device	Holding Register: Allow simultaneous read and write using Modbus instruction 23.
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 by device	UNICODE String value (Holding Register): Reads and writes Unicode Strings using consecutive Holding Registers
STUS	**`STUS:0`**	Varies by device	UNICODE String value (Holding Register): Reads and writes Unicode Strings using consecutive Holding Registers with Byte Swap.
FILE	**`FILE:1:1:10:DumpFile.csv`**	Varies by device	Read File Records command.

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`** and **`STS`** use the following syntax:

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

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

   For **`ST`**, **`STS, STU and STUS`** registers *only*, use the following syntax:

   `<AddressOffset>:<Length> or`

   `<AddressOffset>.[StartByte]:<Length>`

Example — `10:5`

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.
- ***[StartByte]*** (optional): Use this parameter only for **ST** and **STS**, to indicate the initial byte.
- ***<Length>***: Length of the string (in bytes) to be read or written.

➲ **Attention:**

- Use Bit Write commands in the Holding Register for the **Write on Tag Change** field only.
- 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.
- Lastly, keep in mind that when using the **Write Trigger** feature, the driver writes to the entire block of registers from the first address through the last. If there is a register that has not been declared in the worksheet, and its address is within the block, then the register will receive a zero (0) value. Check the worksheet for holes in the address range.
- When an unsigned DWord register type is specified, it can only be associated with a Real database tag in Studio.
- Writing bit values to the **4x** registers is allowed, however there is no such function the Modbus protocol. In this case, the driver performs the *read-mask*-write operation, before writing, the driver first reads the entire word, modify the bit that will be written and then writes back to PLC. Even with this being a very fast operation, if in this period between reading and writing the PLC value changes, the driver will overwrite it with the value that is being processed.
- 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.

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`

Device Register	Header	Address
30001	**3x:1**	0
30010	**3x:0**	10
31020	**3x:1000**	20
30001 and 30001	**FP3:0**	1
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 to 40004	**DF:0**	1
30001 to 30004	**DF3:1**	0
41022 and 41021	**FPS:1000**	21
40001 and 40002	**DW:1**	0
40013 and 40014	**DW:0**	13
41021 and 41022	**DW:1000**	21
40002 and 40001	**DWS:1**	0
31021 and 31022	**DW3:1000**	21
30011 and 30010	**DW3S:1**	10
40001 and 40002	**DWSW:1**	0
40001 and 40002	**BCD:1**	0
31021 and 31022	**BCD3:1000**	21
40001 and 40002	**BCDDW:1**	0
41011 and 41010	**BCDDWS:1000**	10
30001 and 30002	**BCDDW3:1**	0
30011 and 30010	**BCDDW3S:0**	10
40001	**ST:1:2**	0
41010 and 41011	**ST:1000:4**	10
40001 (String 2 chars long)	**STS:1**	0:2
String from 41011 up to 41020, with byte swap	**STS:1000**	10:20

Device Register	Header	Address
40001 to 40004	**DFS:0**	**1**
30001 to 30004	**DF3S:1**	**0**
40001	**STU:0**	**1:10**
40010	**STUS:0**	**10:4**

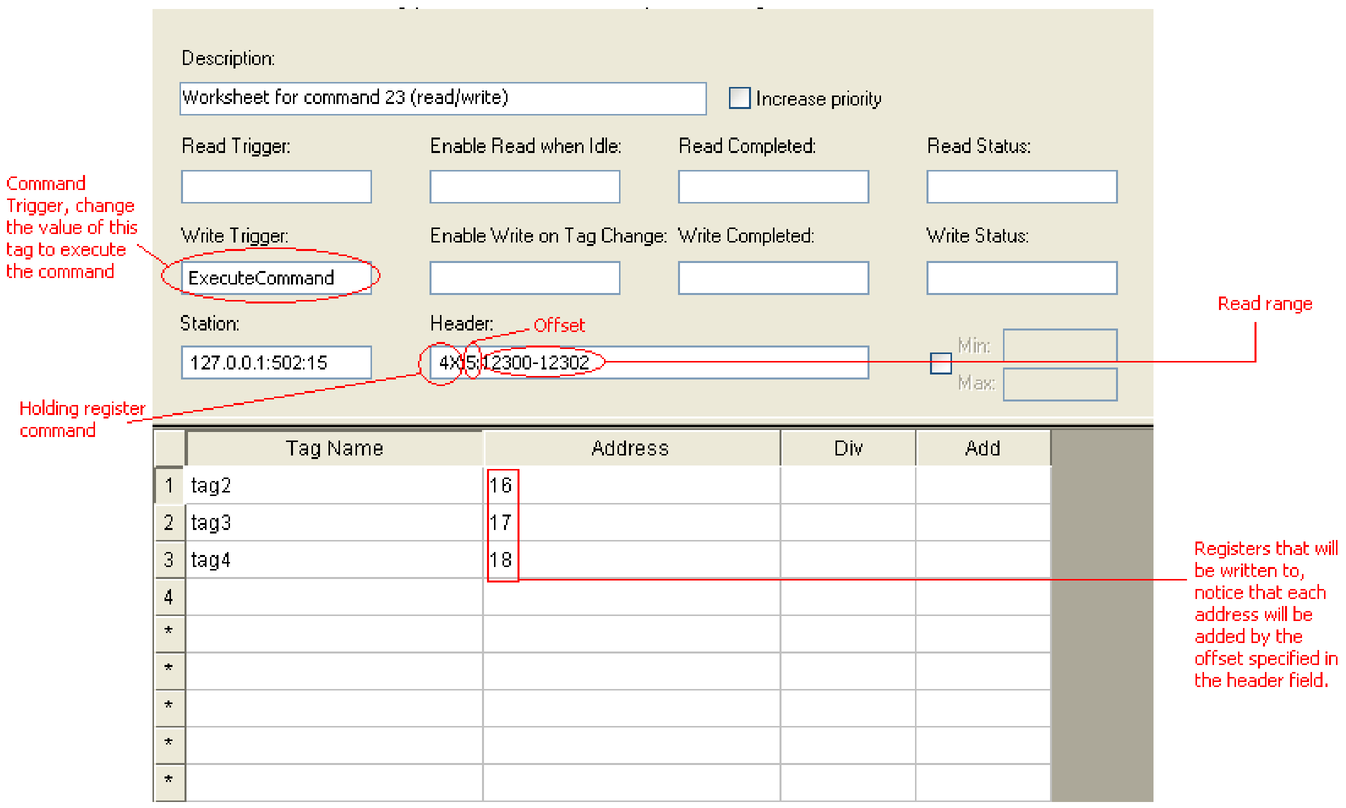
For more information about the device registers and addressing, please consult the manufacturer's documentation.

➲ **Attention:**
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.

Configuring driver sheets for command 23 (simultaneous read/write of holding registers)

The MOTCP driver supports simultaneous reading and writing of holding registers, which can be useful on some specific applications. This command is only supported in the Standard Driver Sheets and it should be configured following the steps below:

1) Configure a Standard Driver Sheet with a holding register header type (4X, FP, FPS, DW, DWS, BCDDW and BCDDWS) header. This will be the write worksheet and it requires an additional range of reading registers at the end.
2) Associate all the addresses that you want to write with their respective tags as you would do when using regular command for Holding Registers.
3) For the addresses that you want to read you need to specify the range in **Header** field using the following format:<Header Type>:<Offset>:<first read address>-<last read address>.
4) Configure a tag in the write trigger field of the worksheet created on step 1. By issuing changing the tag on this field command will be executed. The following picture shows an example of configuration:


Command Trigger, change the value of this tag to execute the command
Offset
Read range
Holding register command
Registers that will be written to, notice that each address will be added by the offset specified in the header field.

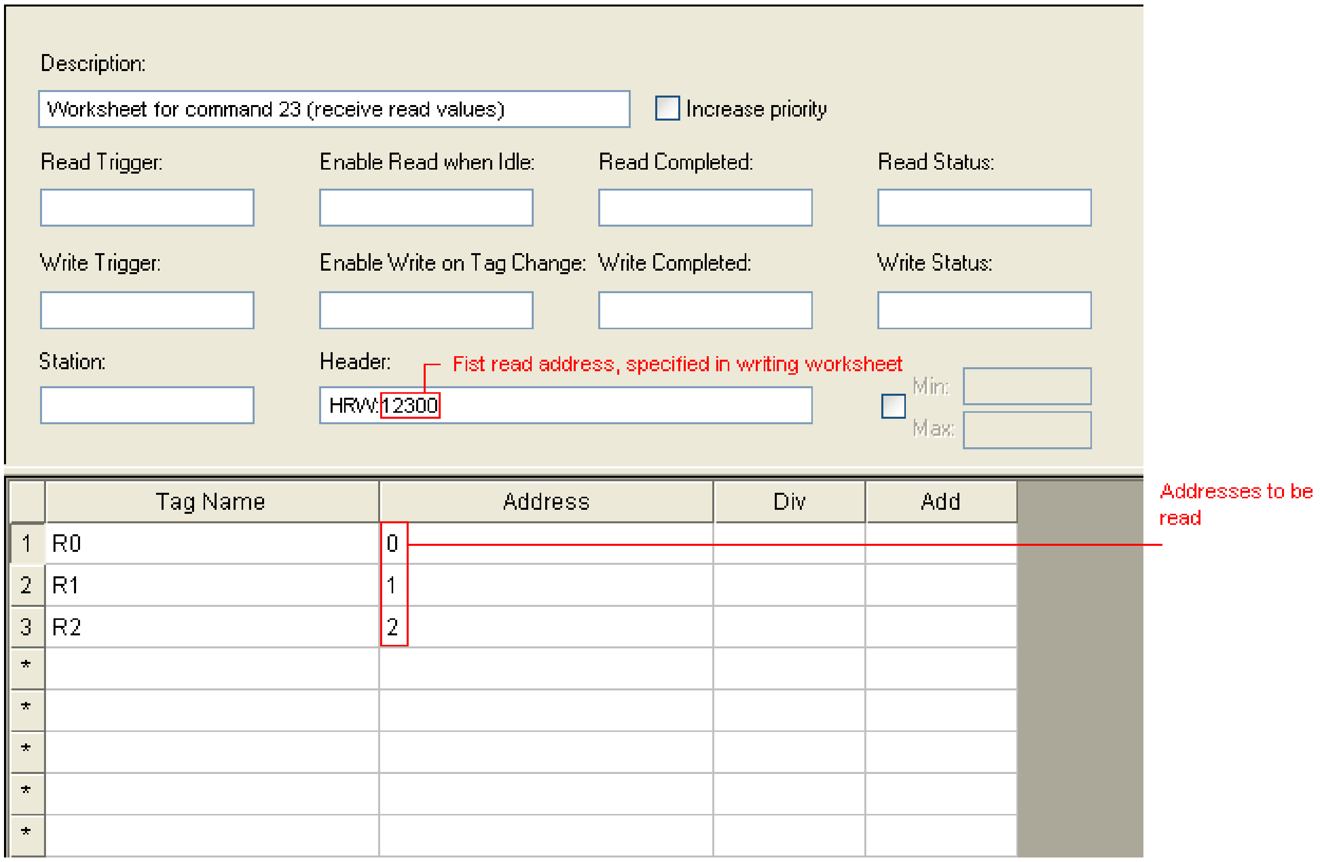
Description: Worksheet for command 23 (read/write)

Increase priority: ☐

Read Trigger:	Enable Read when Idle:	Read Completed:	Read Status:
Write Trigger: ExecuteCommand	Enable Write on Tag Change:	Write Completed:	Write Status:
Station: 127.0.0.1:502:15	Header: 4X:5:12300-12302	Min:	Max:

	Tag Name	Address	Div	Add
1	tag2	16		
2	tag3	17		
3	tag4	18		
4				
*				
*				
*				
*				
*				

5) Create another worksheet that will receive the read values, this worksheet should have the header HRW:<*first read address*>, where *first read address* is the value specified in the **Header** field for the worksheet created on step 1.
6) Associate all the addresses that you are reading with their respective tags, the picture below shows how the worksheet would be configured considering the previous command worksheet example:



Description: Worksheet for command 23 (receive read values)

Increase priority

Read Trigger:

Enable Read when Idle:

Read Completed:

Read Status:

Write Trigger:

Enable Write on Tag Change:

Write Completed:

Write Status:

Station:

Header: HRW:12300

	Tag Name	Address	Div	Add
1	R0	0		
2	R1	1		
3	R2	2		
*				
*				
*				
*				
*				

The addresses in this worksheet can have a data type prefix. The following data types are supported:

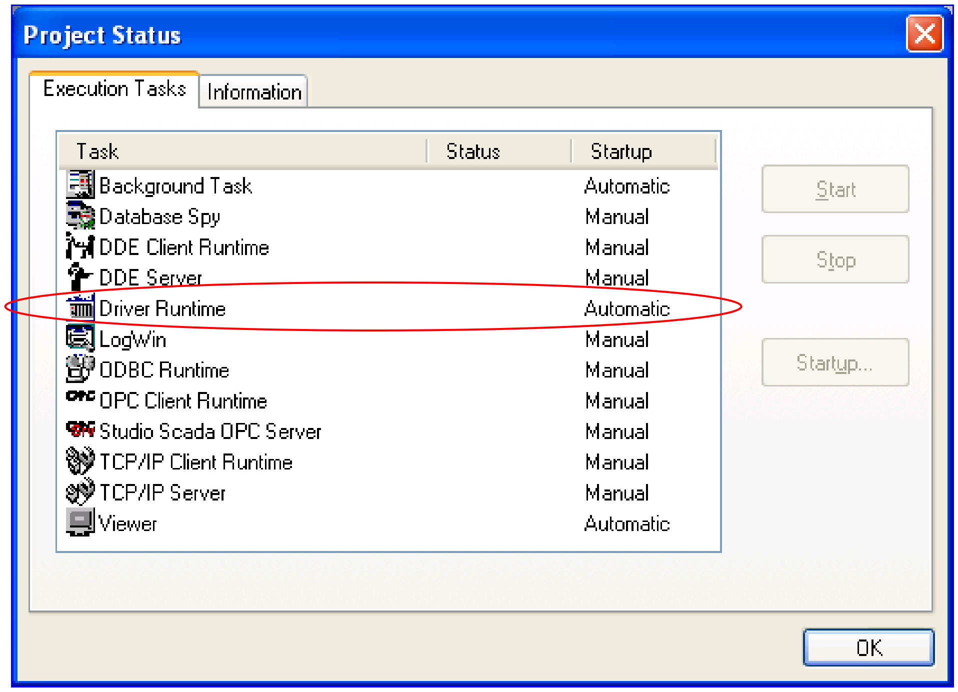
Prefix	Data Type
B	Signed 8 bits variable (byte)
UB	Unsigned 8 bits variable (byte)
W	Signed 16 bits variable (word)
SW	Signed 16 bits variable (word) with byte swap
UW	Unsigned 16 bits variable (word)
USW	Unsigned 16 bits variable (word) with byte swap
DW	Signed 32 bits variable (double word)
SDW	Signed 32 bits variable (double word) with byte swap
UDW	Unsigned 32 bits variable (double word)
USDW	Unsigned 32 bits variable (double word) with byte swap
F	32 bits float points (float)
SF	32 bits float points with byte swap (float)
DF	64 bits float points (double)
SDF	64 bits float points with byte swap (double)
BCD	16 bits BCD value
BCDDW	32 bits BCD value
S	String (address should be S<register>.<Bytes>)

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 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 MOTCP 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 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	ILLEGAL FUNCTION CODE	Function code used by driver is not supported by device.	Check with device manufacturer to support the function code.
2	ILLEGAL DATA ADDRESS	There is an invalid address in the *Driver* worksheet. The data address received in this query was not an allowable address for the slave.	Type a valid address.
3	ILLEGAL DATA VALUE	Specified address contains an invalid value that is not allowed for the slave	Check the address to be sure it exists on the device.
4	SLAVE DEVICE FAILURE	An unrecoverable error occurred while the slave was attempting to perform the requested action.	Check the equipment state. Try rebooting it.
5	ACKNOWLEDGE	Error in the communication Ack action. Specialized use in conjunction with programming commands.	Check the communication parameters for the device and the Studio software.
6	SLAVE DEVICE BUSY	Invalid command if the equipment is in use. Specialized use in conjunction with programming commands.	Studio commands never generate this error.
7	Negative Ack	Error in the communication Ack action. Specialized use in conjunction with programming commands.	Check the communication parameters for the device and the Studio software.
8	Memory parity error	Incorrect communication parameter configuration. Specialized use in conjunction with function codes 20 and 21, to indicate that the extended file area failed to pass a consistency check.	Check configuration of the driver's communication parameters.
10	Invalid Header field	Invalid tag value in the Header field	Specify a valid tag value in the Header field.
11	Invalid Address field	Invalid Address	▪ Check the initial address in the *Driver* worksheet. ▪ Check the Holding register in the *Driver* worksheet with bit configuration. This parameter cannot execute write triggers—it executes "Write on Tag Change" only. ▪ Retype the address in the *Driver* worksheet.
12	Invalid Block size	Offset is greater than the maximum allowed. The maximum offset is usually 64.	Specify a valid offset or create a new *Driver* worksheet.
13	Checksum error	Protocol error	▪ Communication failures that can be cause by electric interferences which invalidated the data sent by the PLC ▪ Generate a Log File and contact your Studio technical support representative
14	Generic TCP/IP error	Wrong TCP/IP network configuration	Use "Telnet" or "Ping" tools to check your network configuration, and try to find the PLC with the computer on which you are running Studio.
15	Invalid IP number	Wrong IP Address	You must specify the IP Address using the appropriate syntax—four fields, a value up to 255 in each field, separated by periods.

Error Code	Description	Possible Causes	Procedure to Solve
16	Connect error	Wrong IP Address or Port Number	Use "Telnet" or "Ping" tools to check your network configuration and try to find the PLC with the computer on which you are running Studio.
18	Invalid BCD Value	Tried to Read an invalid BCD value.	Check the value on PLC to be sure it is a valid value.
19	Invalid BCD Value	Tried to Write a negative BCD value.	Only positive BCD values are valid.
20	Invalid Word Swap	The Swap for Word is not possible	Don't use Word Swap for Strings.
100	Invalid Operation	Writing attempt in the Input Registers, Input Status, or Report Slave ID	You cannot write in these addresses.
1005	Timeout Error	The PLC failed to issue a response to the driver request within the timeout specified in the driver communication settings.	Check if you can "ping" the PLC If you can ping, check if you can "telnet" the Modbus PLC TCP/IP port Check the "Slave ID number" to see if it matches with the value configured in the driver "Station" field Check cable wiring Check the PLC state – in some devices, it must be in RUN mode.
-15	Timeout Start Message	The PLC failed to issue a response to the driver request within the timeout specified in the driver communication settings.	Check if you can "ping" the PLC If you can ping, check if you can "telnet" the Modbus PLC TCP/IP port Check the "Slave ID number" to see if it matches with the value configured in the driver "Station" field Check cable wiring Check the PLC state – in some devices, it must be in RUN mode.
-17	Timeout between rx char	An incomplete response arrived from the PLC	Communication failures that can be cause by electric interferences which invalidated the data sent by the PLC
24	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
25	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 **Protocol Analyzer,** right-click in the *Output* window and select the desired options from the pop-up menu.

You can also use the *Remote LogWin* module to establish an event log on a remote unit that runs on a remote runtime, including Windows CE and Embedded

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 information generated by the command **Support Information** in Studio's **Help** menu ready.

Revision History

Doc. Revision	Driver Version	Author	Date	Description of changes
A	1.01	Sergio A. Poon	Nov/5/1999	Driver available for Windows CE
B	1.02	Sergio A. Poon	Nov/9/1999	Included Protocol Type (default) = PROTOCOL RTU
C	1.03	Sergio A. Poon	Apr/28/2000	Included Floating-point function
D	1.04	Roberto V. Junior	May/4/2000	Fixed bug with Floating-Point function
E	1.05	Roberto V. Junior	Jan/10/2001	Included MAIN DRIVER SHEET feature
F	1.06	Roberto V. Junior	Apr/3/2001	Enhanced performance
G	1.07	Lourenço Teodoro	Jun/20/2001	Enabled TCP/IP time out configuration
H	1.08	Lourenço Teodoro	Jul/23/2001	Fixed bug with write group commands
I	1.09	Lourenço Teodoro	Aug/6/2001	Implemented Signed/Unsigned option by address
J	1.09	Fabíola Fantinato	Dec/5/2001	Revision to conform to documentation standards
K	1.10	Luis F. Rodas	Dec/6/2001	▪ Fixed bug of wrong received values ▪ Fixed bug of Main Driver Sheet
L	1.11	Roberto V. Junior	Jan/8/2002	▪ Fixed bug of address equal zero in the Main Driver Sheet ▪ Included bit read/write command to Holding Register
M	2.00	Eric Vigiani	Jun/27/2002	Modified driver algorithm to avoid Time-out errors due to the station number and message size
N	2.01	Eric Vigiani	Jul/22/2002	Included DW data type
O	2.02	Eric Vigiani	Aug/05/2002	Fixed problem when writing DW blocks
P	2.03	Eric Vigiani	Aug/26/2002	Removed ASCII protocol type from the *Communication Parameters* dialog window
Q	2.04	Eric Vigiani	Sep/30/2002	Modified DW writing algorithm to execute command 10 instead of command 06
R	2.05	Fabio H.Y.Komura	Jan/07/2003	Included FP3 and FP3S Headers (Read Float Point to Input Register)
S	2.06	Eric Vigiani	Aug/22/2003	Included FPS data type
T	2.07	Eric Vigiani	Dec/18/2003	Included DWS data type
U	2.08	Lourenço Teodoro	Mar/16/2004	Implemented disconnection after time out to avoid problems with devices that do not accept multiple messages (the driver was not sending multiple messages; however, the transport layer was doing it while the socket was open)
V	2.09	Eric Vigiani	May/25/2004	Implemented writing group commands when writing FP values
W	2.10	Fábio H.Y. Komura	Jun/16/2004	▪ Added DW3 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)
X	2.11	Eric Vigiani	Jul/02/2004	Fixed problem with bit addressing on the MAIN DRIVER SHEET.
Y	2.12	Fabio H. Y. Komura	Sep/03/2004	Added support to BCD, BCD3, BCDDW, BCDDWS, BCDDW3 and BCDDW3S data type.
Z	2.13	Bruno A. Crepaldi	Nov/12/2004	Fixed problem with 1x (Input Status) (Read).
AA	2.14	Fabio H. Y. Komura	Dec/09/2004	Changed the **Swap** parameter to comply with old versions of the MODBU driver (version 2.10 or older).
AB	2.15	Leandro Coeli	Jan/18/2005	Added support to String type

AC	2.16	Leandro Coeli	Feb/18/2005	Fixed communication problem
AD	2.17	Leandro Coeli	Apr/26/2005	Implemented configurable Block Size
AE	2.18	Leandro Coeli	Jan/20/2006	Fixed problem on ST Header
AF	2.18	Michael D. Hayden	Jun/14/2006	Edited for language and usability.
AG	2.19	Graziane C. Forti	Aug/31/2006	Implemented the Unsigned in the all headers. Implemented Block size 1 for FP or DW type in the Main Driver Sheet. Added support for STS (String with Byte Swap) register type.
AH	2.20	Arthur S. Allievi	Nov/10/2006	Signed or Unsigned choice restricted to some drivers Fixed syntax problems with .Len and .Bit Fixed Block size problems Fixed Write Group issue Changed the Configuration Parameters Dialog. Image updated.
AI	2.21	Graziane C. Forti	Nov/23/2006	Fixed problem with writing of bit Modified to compile for WinCE Fixed the String length configuration Fixed the String configuration in the SDS Fixed the String writing
AJ	2.22	Plínio M. Santana	Jan/18/2007	Fixed problem about invalid addresses (negatives).
AK	2.23	Plínio M. Santana	Jul/06/2007	DF and DF3 types created. Document updated.
AL	2.24	Plínio M. Santana	Jan/30/2008	Modification to send the correct error message Fixed the virtual read groups
AM	2.24	Plínio M. Santana	Apr/2/2008	Fixed document on errors table (error code #4).
AN	10.1	Marcelo Carvalho	Jan/07/2009	Updated driver version, no changes in the contents.
AO	10.1	Andre Bastos	Apr/10/2009	Modified documentation only
AP	10.3	Lourenço Teodoro	Jul/1/2009	Created section for command 23 Added new parameter MaxGap
AQ	10.4	Andre Korbes	Sep/16/2010	Fixed bug when connecting to same IP but different ports
AR	10.5	Paulo Balbino	Jan/17/2013	Added the capability of configuring the Invocation Identifier byte as a fixed or incremental value
AS	10.6	Paulo Balbino	Aug/15/2013	Fixed bug with size of request Modified to allow bit and string writes using the write trigger Fixed problem of requesting more WORD with ST and STS datatypes
AT	10.7	Charan Manjunath P	Oct/21/2013	Updated Error Codes table with Error Code 1 and removed Error Code 17
AU	10.8	Charan Manjunath P	Nov/01/2013	Updated with FPSW in the list of data types and address ranges
AV	10.9	Priya Yennam	Jan/20/2014	Updated with DWSW in the list of data types
AW	10.10	Charan Manjunath P	Mar/5/2014	Fixed issue of getting the error code on write operations.
AX	10.11	Felipe Andrade	Oct/17/2014	Updated with FP3SW in the list of data types Added support for DFS and DF3S.
AY	10.12	Paulo Balbino	Dec/2/2014	Fixed issue with “invalid address” error when accessing individual bits from 3x and 4x addresses during the runtime
AZ	10.13	Priya Yennam	Jan/30/2015	Added support to UNICODE strings
BA	10.14	Anushree Phanse	May/05/2015	Fixed the issue of not showing an error code while reading or writing in case of error
BB	10.15	Anushree Phanse	May/26/2015	Fixed Problem with Validating function received that incorrectly caused driver to

				allow invalid 'OK' response and generate no error. Changed Documentation to show that driver allows PLC ID from 1 - 254
BC	10.16	Anushree Phanse	Feb/24/2016	Added validation for writing group functions
BD	10.17	Priya Yennam	Sept/09/2016	Added Read File Records Command
BE	10.18	Anushree Phanse	Jan/09/2017	Fixed invalid block size issue for strings.
BF	10.18	Anushree Phanse	Jun/08/2017	Changed documentation to have accurate information about using indirect tags on MDS. No change in the driver.
BG	10.18	Eduardo Castro	Aug/24/2017	The list of data type prefixes was changed in this documentation. No changes in the driver.
BH	10.19	Anushree Phanse	Jan/16/2019	Fixed the driver to correctly interpret some TCP TX messages.