MODSL Communication Driver

Driver (Slave) for Serial and Ethernet Communication
with Devices Using the Modbus Protocol

Contents

Introduction

The MODSL driver (slave) enables Serial or Ethernet 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 MODSL 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 MODSL driver in the Studio system.
- **Configuring the Driver**: Explains how to configure the MODSL driver in the Studio system, including how to associate database tags with device registers.
- **Executing the Driver**: Explains how to execute the MODSL 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 MODSL 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/7/8 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 MODSL driver (slave) 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:

- **Compatible Equipment:** Any device or program that communicates using the Modbus protocol for Serial or Ethernet communication in *Master* mode
- **Programmer Software:** None specifc

Network Specifications

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

- **Physical Protocol**: Serial (RS232/485) or Ethernet (TCP/IP)
- **Logic Protocol**: Modbus
- **Device Runtime Software**: None
- **Specific PC Board**: None

Driver Characteristics

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

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

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

- Windows 7/8/10 and Servers
- Windows CE

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

The MODSL 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	●	●	●	●	●	●	●	●	●

Conformance Testing

The following hardware/software was used for conformance testing:

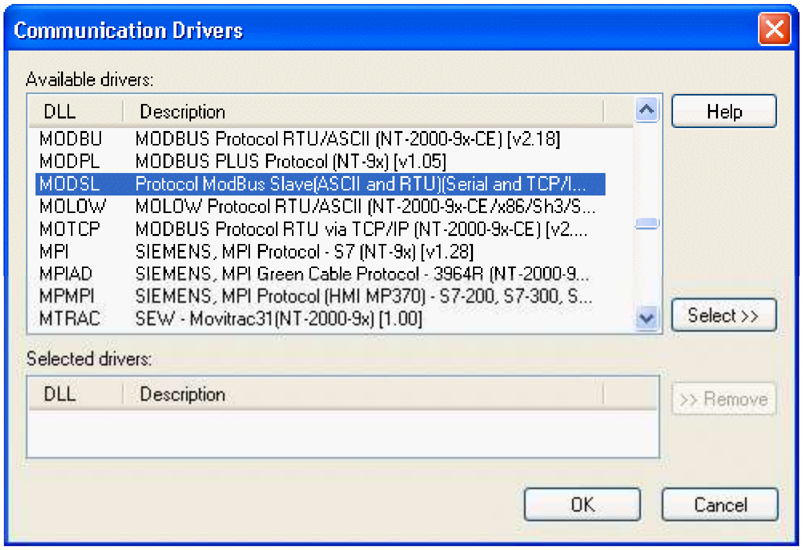
- **Configuration (Serial)**:
  - **PLC Program**: None
  - **Baud Rate**: 9600
  - **Protocol**: RTU/ASCII
  - **Data Bits**: 8/7
  - **Stop Bits**: 1
  - **COM Port**: COM1
- **Configuration (Ethernet)**:
  - **PLC Program**: None
  - **Baud Rate**: Not used
  - **Protocol**: RTU
  - **Data Bits**: Not used
  - **Stop Bits**: Not used
  - **COM Port**: Not used
  - **Port Number**: 502
- **Cable**: Use specifications described in the “Network Specifications” section above.

Driver Version	Studio Version	Operating System (development)	Operating System (target)	Equipment
3.3	8.1 + SP2	Windows 8	Windows 8 x64 Windows 7 x64 Windows CE 7.0 ARMV4I Windows CE 5.0 x86 Windows CE 7.0 x86	Modbus master driver (MODBU) – Serial Modbus master driver (MOTCP) – TCP/IP Running on Windows PCs and CE devices. Modbus Master compatible equipment.

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 MODSL driver for your Studio application:

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



Available drivers:

DLL	Description
MODBU	MODBUS Protocol RTU/ASCII (NT-2000-9x-CE) [v2.18]
MODPL	MODBUS PLUS Protocol (NT-9x) [v1.05]
MODSL	Protocol ModBus Slave(ASCII and RTU)(Serial and TCP/I...
MOLOW	MOLOW Protocol RTU/ASCII (NT-2000-9x-CE/x86/Sh3/S...
MOTCP	MODBUS Protocol RTU via TCP/IP (NT-2000-9x-CE) [v2....
MPI	SIEMENS, MPI Protocol - S7 (NT-9x) [v1.28]
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]

Selected drivers:

DLL	Description

Communication Drivers Dialog

3. When the **MODSL** 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 Driver

Once you have selected the MODSL 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.

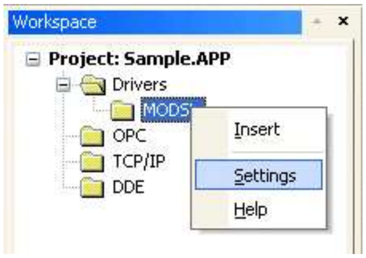
Configuring the Communication Settings

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.

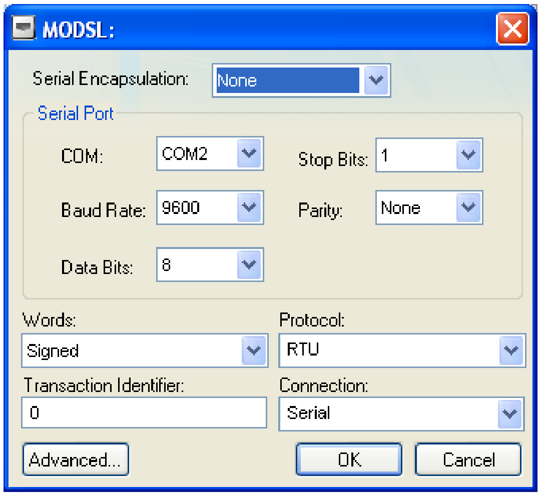
For the purposes of this document, only MODSL driver-specific settings and procedures will be discussed here. To configure the communication settings for the MODSL driver:

1. In the *Workspace* pane, select the *Comm* tab and then expand the *Drivers* folder. The MODSL driver is listed here as a subfolder.

2. Right-click on the *MODSL* subfolder and then select the **Settings** option from the pop-up menu. The *MODSL: Communication Parameters* dialog is displayed:



Select Settings from the Pop-Up Menu



Serial Encapsulation: None

Serial Port

COM: COM2

Stop Bits: 1

Baud Rate: 9600

Parity: None

Data Bits: 8

Words: Signed

Protocol: RTU

Transaction Identifier: 0

Connection: Serial

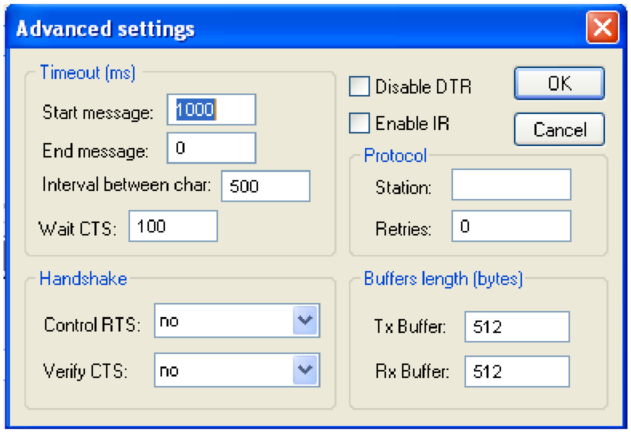
MODSL: Communication Parameters Dialog

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
COM	COM2	COM1, COM8	Serial port of the PC used to communicate with the device.
Baud Rate	9600	110 to 57600	Communication rate of data.
Data Bits	8	5 to 8	Number of data bits used in the protocol. (ASCII is typically 7 bits; RTU is typically 8 bits.)
Stop Bits	1	1 or 2	Number of stop bits used in the protocol.
Parity	None	Even, Odd, None, Space or Mark	Parity of the protocol.
0-Signed Value > 1-Unsigned Value	0	0	Values are unsigned.
		1	Values are signed.
Protocol	RTU	ASCII	Each eight-bit Word is sent as two four-bit ASCII characters, allowing for a time interval between characters without causing errors. (ASCII protocol does not work with TCP/IP communication)
		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.
Transaction Identifier	0	0	Do not use Transaction Identifier.
		1	Do use Transaction Identifier.
Connection	S	S	Serial communication.
		T	Ethernet TCP/IP communication.

5. 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

Advanced Settings Dialog

When the dialog is displayed, configure the **Station** setting in the following format.

<Slave ID>*:*<optPortNumber>

Where

<Slave ID> is slave number (**1** to **99**) of this Modbus Device in the Modbus Network
<optPortNumber> is an optional parameter for the TCP port number that the driver will open and keep listening to. If this value is omitted, the driver will use the default value which is **502**

6. 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. 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. IMPORTANT: If you are using Windows 95/98 or Windows CE with the correct RS232/RS485 adapter (i.e. without RTS control), then you must select this option.
		yes	Set the RTS (Request to Send) handshake signal before communication. IMPORTANT: If you are using Windows NT and the Cutler-Hammer RS232/RS485 adapter, then you must select this option.
		yes+echo	Set the RTS (Request to Send) handshake signal before communication, and echo the signal received from the target device.

➲ **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.

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

Configuring the Driver Worksheets

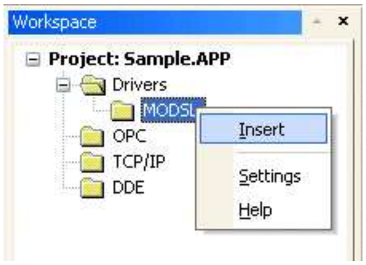
A selected driver includes one or more driver worksheets, which are used to associate database tags in Studio with operands on the target device. Each worksheet is triggered by specific application behavior, so that the tags / operands defined on that worksheet are scanned only when necessary – that is, only when the application is doing something that requires reading from or writing to those specific tags / operands. Doing this optimizes communication and improves system performance.

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.

To insert a new driver worksheet:

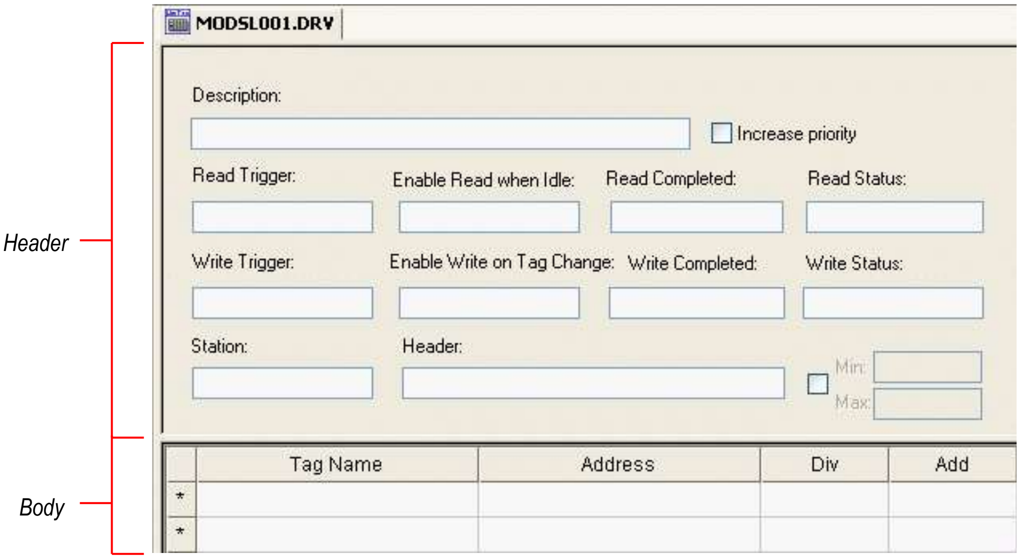
1. In the *Comm* tab, open the *Drivers* folder and locate the *MODSL* subfolder.

2. Right-click on the *MODSL* subfolder, and then select **Insert** from the pop-up menu:



Inserting a New Worksheet

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



MODSL Driver Worksheet

> **Note:**
>
> Worksheets are numbered in order of creation, so the first worksheet is **`MODSL001.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 MODSL driver.

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

   - **Station** field: Not used.

   - **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:1000**

Where:

– ***<Type>*** is the register type (**0X**, **1X**, **3X**, **4X**, **FP**, **DW, ST, STS, STU or STUS**).

– ***<AddressReference>*** is the initial address (reference) of the configured type, must be multiple of 1000.

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.
3X	**3X:0**	Varies according to equipment	Input register: Reads events using Modbus instruction 04.
FP3	**FP3:0**	Varies according to equipment	Floating-point value (Input Register): Reads floating-point values using two consecutive Input Registers.
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.
DW	**DW:0**	Varies according to equipment	DWord value (Holding Register): Reads and writes DWord values using two consecutive Holding Registers.
DF	**DF:0**	Varies according to equipment	Long Real or Double Floating Point (Holding Register): Reads and writes floating point values using 4 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.
ST	**ST:0**	Varies according to equipment	String values (Holding Registers): Reads and writes strings for the Holding Registers
STS	**STS:0**	Varies according to equipment	String values with byte swap (Holding Registers): Reads and writes strings with bytes swap within registers for Holding Registers.
STU	**STU:0**	Varies according to equipment	Unicode Strings (Holding Registers): Reads and writes UNICODE strings for holding registers.

Data Types	Sample Syntax	Valid Range of Initial Addresses	Comments
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 Holding Registers.

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

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

   Examples — `10`, `S20`, `U40`, `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.

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

   `<AddressOffset>.<Length>`

   Where:

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

   Example — **`ST:10.5, STU:1.10`**

➲ **Attention:**

- The Floating-point (header FP) 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.
- You can have up to 1000 addresses on the same worksheet

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`

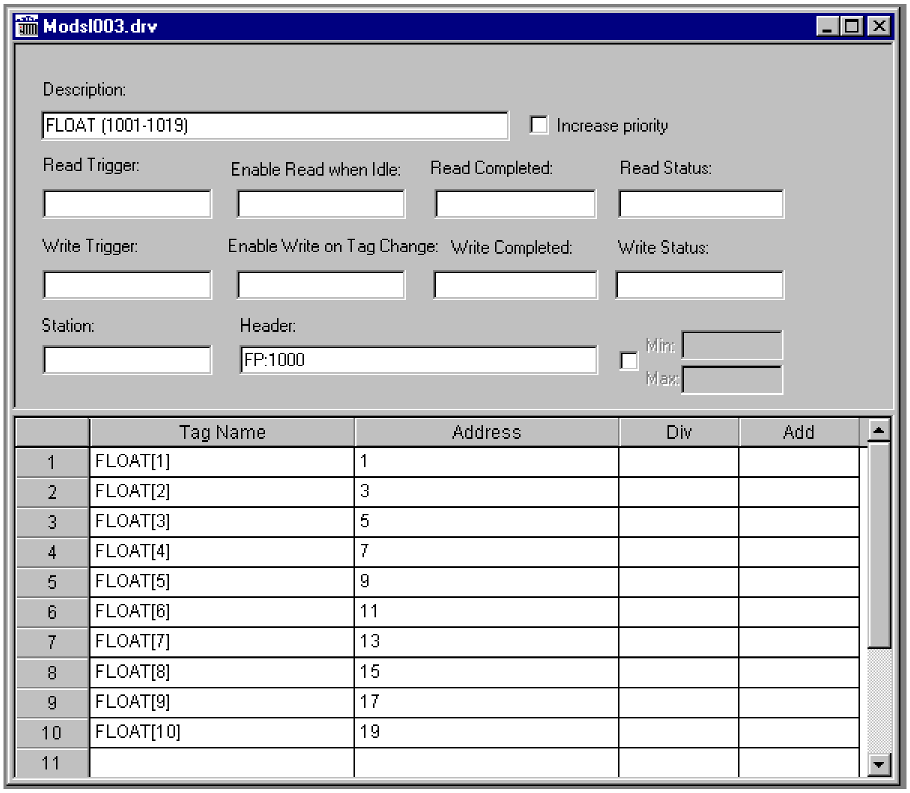
Device Register	Header	Address
30001	3X:1	0
30010	3X:0	10
31020	3X:1000	20
40001	4X:1	0
40010	4X:0	10
41020	4X:1000	20
40010 (bit 0)	4X:0	10.0
41010 (bit 7)	4X:1000	10.7
40001 and 40002	FP:0	1
40013 and 40014	FP:0	13
41021 and 41022	DW:1000	21
40010	ST:0	10.6
40120	STS:100	20.8
40230	STU:200	30.10
40040	STUS:0	40.4

➲ **Attention:**

- The Headers must be configured with the offset 0 or multiples of 1000 (e.g.: 4x:0, FP:1000, DW:2000, etc) for multiple headers to work in the same application. This validation is performed when the header is filled.
- The Address field (from the body of the worksheet) cannot be configured with a value higher than 999.
- You cannot have more than one worksheet with the same Header. Otherwise, the communication will not work properly.
- If the remote Modbus Master device requests an address that is not configured in the driver worksheet, the value 0 (zero) will be sent by the MODSL driver if there is a driver sheet with a matching header.

`FP` and **`DW`** are special types; worksheets configured using these types must have all of their Address values be either odd or even, but not a mixture of both. See the following illustrations for examples of correctly and incorrectly configured worksheets.

Example 1 – Correctly configured for floating point odd (41001–41002, 41003–41004, …, 41019–41020):



Description: FLOAT (1001-1019)

□ Increase priority

Read Trigger:

Enable Read when Idle:

Read Completed:

Read Status:

Write Trigger:

Enable Write on Tag Change:

Write Completed:

Write Status:

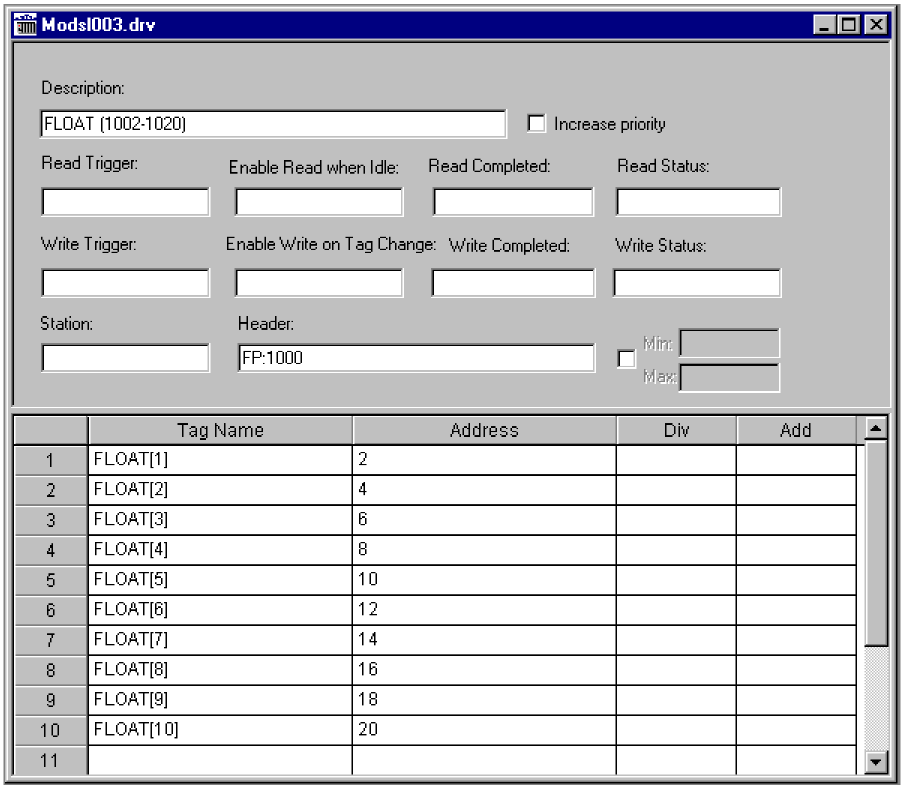
Station:

Header: FP:1000

□ Min: Max:

	Tag Name	Address	Div	Add
1	FLOAT[1]	1		
2	FLOAT[2]	3		
3	FLOAT[3]	5		
4	FLOAT[4]	7		
5	FLOAT[5]	9		
6	FLOAT[6]	11		
7	FLOAT[7]	13		
8	FLOAT[8]	15		
9	FLOAT[9]	17		
10	FLOAT[10]	19		
11				

Example 2 – Correctly configured for floating point even (401002–401003, 401004–401005, …, 401020–401021):



Description: FLOAT (1002-1020)

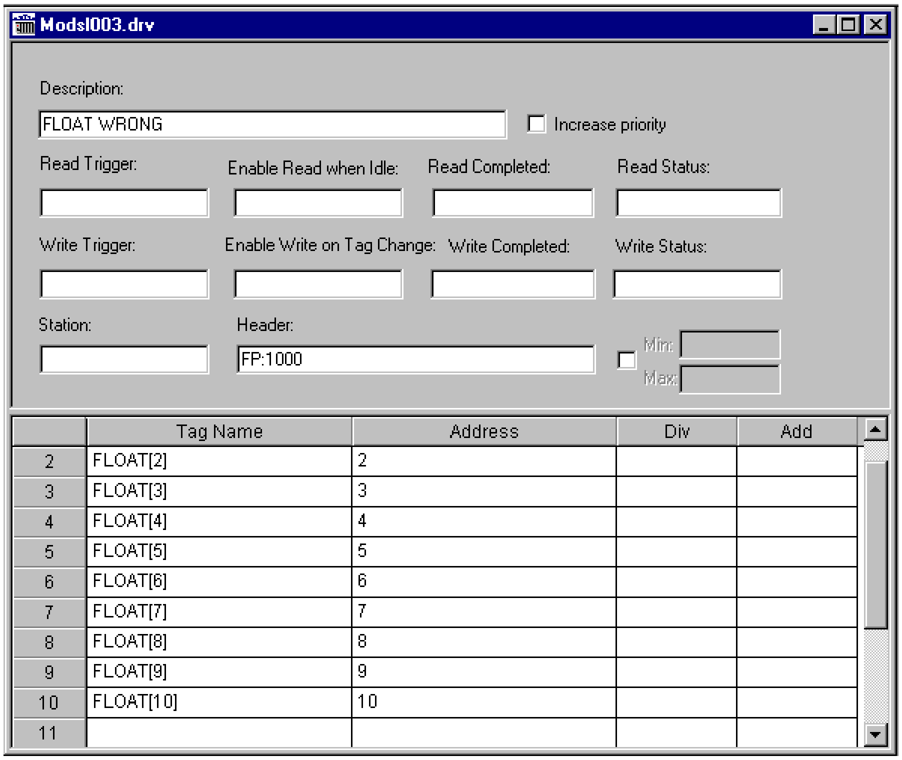
☐ Increase priority

Read Trigger:
Enable Read when Idle:
Read Completed:
Read Status:
Write Trigger:
Enable Write on Tag Change:
Write Completed:
Write Status:
Station:
Header: FP:1000

☐ Min: Max:

	Tag Name	Address	Div	Add
1	FLOAT[1]	2		
2	FLOAT[2]	4		
3	FLOAT[3]	6		
4	FLOAT[4]	8		
5	FLOAT[5]	10		
6	FLOAT[6]	12		
7	FLOAT[7]	14		
8	FLOAT[8]	16		
9	FLOAT[9]	18		
10	FLOAT[10]	20		
11				

Example 3 – Incorrectly configured:



Description: FLOAT WRONG

☐ 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: FP:1000

☐ Min: Max:

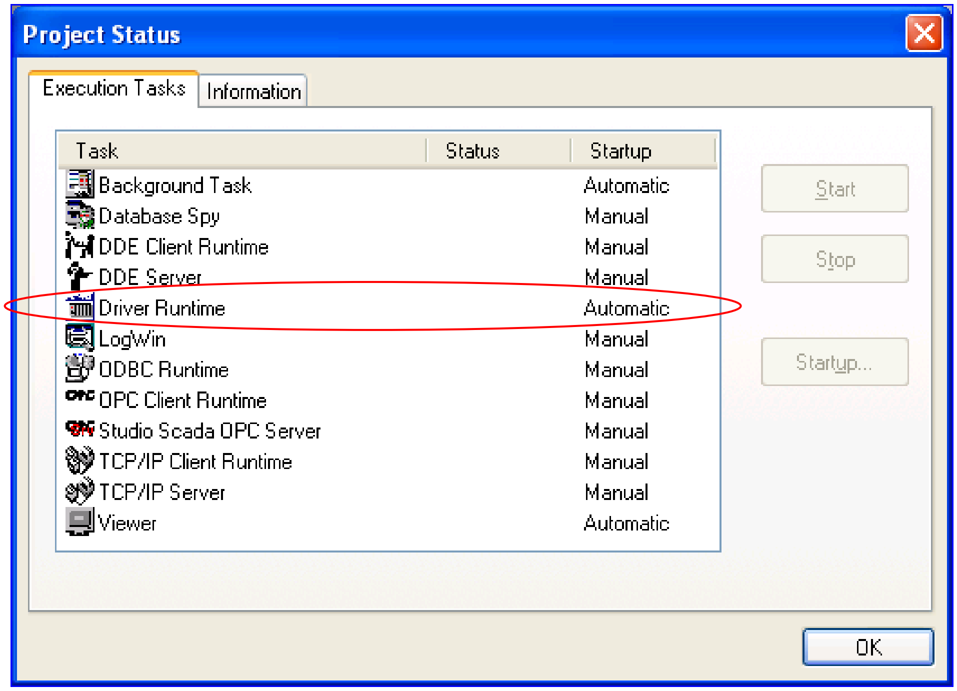
	Tag Name	Address	Div	Add
2	FLOAT[2]	2		
3	FLOAT[3]	3		
4	FLOAT[4]	4		
5	FLOAT[5]	5		
6	FLOAT[6]	6		
7	FLOAT[7]	7		
8	FLOAT[8]	8		
9	FLOAT[9]	9		
10	FLOAT[10]	10		
11				

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 MODSL 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.
2	Illegal data address	Address requested from master is not configured in Studio communication sheets	Create a worksheet with tags matching the requested data.
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	Checksum error	Error in checksum received	Verify the communication parameters (see “Configuring the Communication Settings” for valid configuration).
15	Fail in message received	Unsolicited message could not be processed.	Verify the communication parameters
16	Invalid command received	Invalid command	Drivers (slave) do not allow read/write commands made by the user.
17	Invalid protocol	Invalid protocol	Choose ASCII or RTU protocol.
18	Invalid communication	Invalid communication	Choose **S** for Serial or **T** for TCP/IP communication.

⇨ **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 (Remote **LogWin**) to establish an event log on a remote unit (e.g. that runs Windows CE or XP Embedded).

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 Applications

There is no Sample Application for this driver

Revision History

Doc. Revision	Driver Version	Author	Date	Description of Changes
A	1.00	Lourenço Teodoro	10-Jan-2001	First driver version
B	1.01	Lourenço Teodoro	05-Mar-2002	Inserted the Rx log messages
C	2.00	Rafael	08-Aug-2002	Inserted TCP/IP communication
D	2.01	Eric Vigiani	10-Dec-2003	Included the Transaction Identifier in Communication Parameters
E	2.02	Lourenço Teodoro	03-Mar-2004	Fixed problems with buffer overflow and time outs.
F	2.02	Arthur Allievi	09-Oct-2006	Fixed some problems in the documentation.
G	2.02	Michael D. Hayden	08-Dec-2006	Edited for language and usability.
H	2.03	Rafael R. Fernandes	02-Jul-2007	Station field corrected (only documentation). Added information about ASCII protocol not working with TCP/IP Fixed problem with function 15. (Group writing for Coils)
I	2.04	André Körbes	23-Sep-2010	Fixed support for bits of header 3X and 4X
J	2.5	André Körbes	24-Jun-2011	Improved address validation and documentation.
K	2.6	André Körbes	10-Jul-2012	- Added support for headers FP3 and DF. - Improved error handling and messaging
L	2.7	André Körbes	21-Oct-2013	Fixed problem with tags not receiving the correct bit value.
M	2.8	Priya Yennam	20-Jan-2014	Added the capability of configuring the TCP Port Number in the Advanced Settings.
N	2.9	Charan Manjunath	05-Mar-2014	Fixed issue of writing to 0X and FP headers.
O	2.9	Anoop R	11-Aug-2014	Added support for header DF3.
P	3.0	Priya Yennam	29-Jan-2015	Added String support – ST, STS Added Unicode Strings support – STU, STUS
Q	3.1	Anushree Phanse	30-Nov-2015	Improved driver scalability on PC and WinCE
R	3.2	Anushree Phanse	18-Aug-2016	Fixed the timeout issue between MODSL and MODBU driver..
S	3.3	Anushree Phanse	06-Dec-2018	Added support for RS485 multidrop