

**MODSL Communication Driver**

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

**Contents**

**INTRODUCTION ..... 2**

**GENERAL INFORMATION..... 3**

    DEVICE CHARACTERISTICS ..... 3

    LINK CHARACTERISTICS ..... 3

    DRIVER CHARACTERISTICS ..... 4

    CONFORMANCE TESTING ..... 4

**INSTALLING THE DRIVER ..... 5**

**CONFIGURING THE DEVICE ..... 6**

**CONFIGURING THE DRIVER ..... 6**

    SETTING THE COMMUNICATION PARAMETERS ..... 6

    CONFIGURING THE STANDARD DRIVER WORKSHEET ..... 9

**EXECUTING THE DRIVER..... 14**

**TROUBLESHOOTING ..... 15**

**SAMPLE APPLICATION ..... 16**

**REVISION HISTORY..... 17**

## Introduction

The MODSL driver enables communication between the Studio system and a set of devices using the Modbus protocol slave, according to the specifications discussed in this document.

This document was designed to help you install, configure, and execute the MODSL driver to enable communication with these devices. The information in this document is organized as follows:

- **Introduction:** Provides an overview of the MODSL driver documentation.
- **General Information:** Provides information needed to identify all the required components (hardware and software) used to implement communication between Studio and the MODSL driver.
- **Installing the Driver:** Explains how to install the MODSL driver.
- **Configuring the Device:** Explains how to configure the Modbus device.
- **Configuring the Driver:** Explains how to configure the communication driver.
- **Executing the Driver:** Explains how to execute the driver to verify that you installed and configured the driver correctly.
- **Troubleshooting:** Lists the most common error codes for this protocol and explains how to fix these errors.
- **Sample Application:** Explains how to use a sample application to test the driver configuration.
- **Revision History:** Provides a log of all modifications made to the driver and the documentation.



### Notes:

- This document assumes that you have read the “Development Environment” chapter in the product’s *Technical Reference Manual*.
- This document also assumes that you are familiar with the Windows NT/2000/XP environment. If you are unfamiliar with Windows NT/2000/XP, we suggest using the **Help** feature (available from the Windows desktop **Start** menu) as you work through this guide.

## General Information

This chapter explains how to identify all the hardware and software components used to implement serial communication between the Studio MODSL driver and devices using the Modbus protocol.

The information is organized into the following sections:

- Device Characteristics
- Link Characteristics
- Driver Characteristics

### Device Characteristics

To establish serial communication, you must use devices with the following specifications:

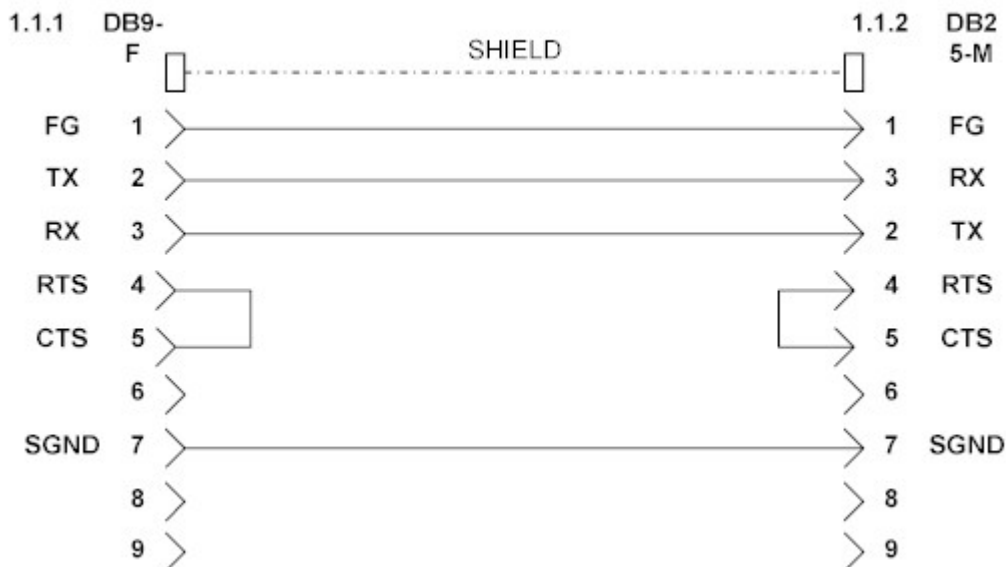
- **Manufacturer:** Any device using the Modbus (master) protocol through Serial or Ethernet communication
- **Compatible Equipment:** Supervisory systems using Modbus master
- **Programmer Software:** ModSoft

For a list of the devices used for conformance testing, see “Conformance Testing” on page 4.

### Link Characteristics

To establish serial communication, you must use links with the following specifications:

- **Device Communication Port:** Modbus Port (AEG 985E)
- **Physical Protocol:** RS232 or TCP/IP
- **Logic Protocol:** Modbus
- **Device Runtime Software:** None
- **Specific PC Board:** None
- **Cable Wiring Scheme:**



**Cable Wiring Scheme**

## **Driver Characteristics**

The MODSL driver is composed of the following files:

- **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**: Document providing detailed information about the MODSL driver.
- **MODSL.DLL**: Compiled driver.

### **Notes:**

- All of the preceding files are installed in the **/DRV** subdirectory of the Studio installation directory.
- You must use Adobe Acrobat® Reader™ (provided on the Studio installation CD-ROM) to view the **MODSL.PDF** document.

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

- Windows CE
- Windows 9x
- Windows 2000
- Windows NT
- Windows XP

For a list of the operating systems used for conformance testing, see the “Conformance Testing” section.

## **Conformance Testing**

The following hardware/software was used for conformance testing:

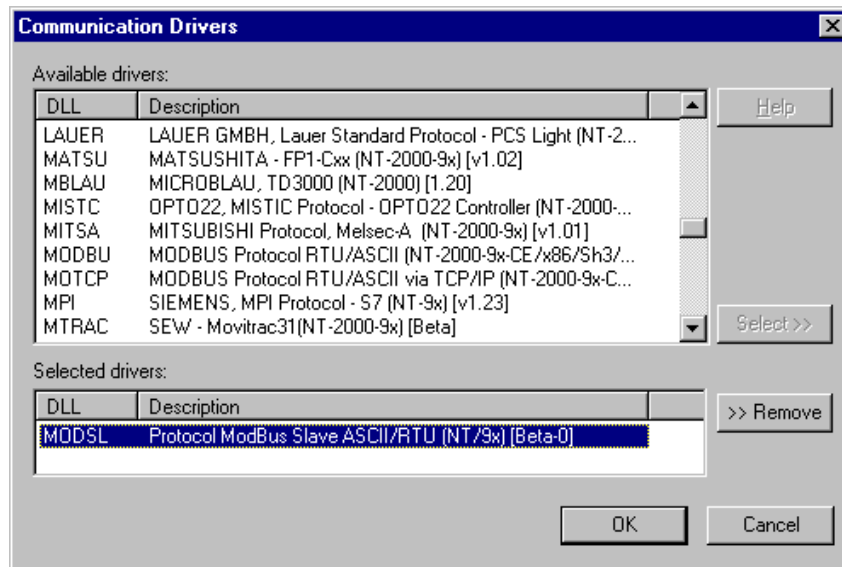
- **Equipment**: PC using Studio with Modbus master driver (MODBU)
- **Configuration (Serial)**:
  - **PLC Program**: None
  - **Baud Rate**: 9600
  - **Protocol**: RTU/ASCII
  - **Data Bits**: 8/7
  - **Stop Bits**: 1
  - **COM Port**: COM1
  - **Port Number**: Not Used
- **Configuration (Ethernet)**:
  - **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 “Link Characteristics” section
- **Operating System (development)**: Windows 2000 Professional
- **Operating System (target)**: Windows 2000 Professional
- **Studio Version**: 5.1
- **Driver Version**: 2.00

## Installing the Driver

When you install Studio version 5.1 or higher, all of the communication drivers are installed automatically. You must select the driver that is appropriate for the application you are using.

Perform the following steps to select the driver from within the application:

1. Open Studio from the **Start** menu or double-click the Studio shortcut icon on your desktop.
2. From the Studio main menu bar, select **File** → **Open Project** to open your application.
3. Select **Insert** → **Driver** from the main menu bar to open the *Communication Drivers* dialog.
4. Select the **MODSL** driver from the *Available Drivers* list, and then click the **Select** button.



*Communication Drivers Dialog*

5. When the **MODSL** driver displays in the **Selected Drivers** list, click the **OK** button to close the dialog.

**Note:**

It is not necessary to install any other software on your computer to enable communication between Studio and the device. However, to download the custom program to your device you must install a Modbus programmer software package (such as *ModSoft*). Consult the Modbus documentation for installation instructions.

**Caution:**

For safety reasons, you must use special precautions when installing the physical hardware. Consult the hardware manufacturer's documentation for specific instructions in this area.

## 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: Even

## Configuring the Driver

After opening Studio and selecting the MODSL driver, you must configure the driver. Configuring the MODSL driver is done in two parts:

- Specifying communication parameters (only one configuration needed).
- Defining communication tags and controls in the Communication tables or *Driver* worksheets (Standard Driver Worksheets).

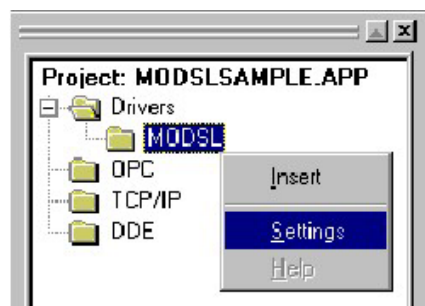
Worksheets are divided into two sections, a *Header* and a *Body*. The fields contained in these two sections are standard for all communications drivers — except the **Station**, **Header**, and **Address** fields, which are driver-specific. This document explains how to configure the **Station**, **Header**, and **Address** fields only.

**Note:**  
For a detailed description of the Studio Standard Worksheets, and information about configuring the standard fields, review the product's *Technical Reference Manual*.

### Setting the Communication Parameters

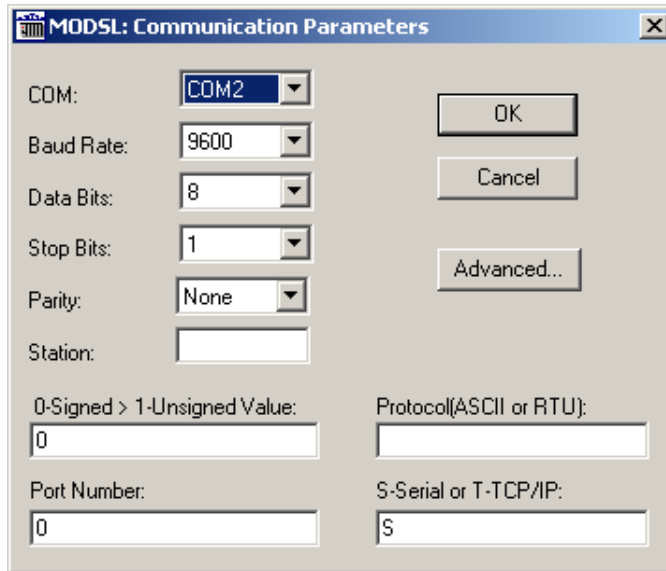
Use the following steps to configure the communication parameters, which are valid for all driver worksheets configured in the system):

1. From the Studio development environment, select the **Comm** tab located below the *Workspace*.
2. Click on the *Drivers* folder in the *Workspace* to expand the folder.
3. Right-click on the *MODSL* subfolder and when the pop-up menu displays, select the **Settings** option:



Select Settings from the Pop-Up Menu

The MODSL: Communications Parameters dialog displays:



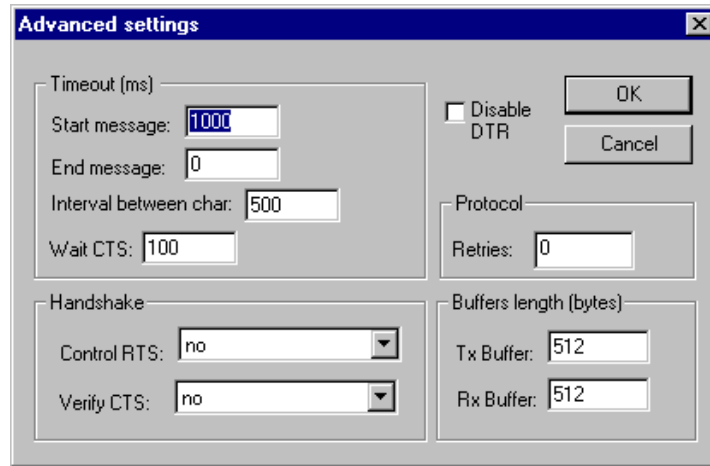
**MODSL: Communication Parameters Dialog**

4. Specify the parameters as noted in the following table:

| Parameters                               | Default Values | Valid Values                   | Description  |
|--|----------------|--------------------------------|--|
| COM                                      | COM2           | COM1 to COM8                   | Serial port of the PC used to communicate with the device                |
| Baud Rate                                | 9600           | 110 to 57600 bps               | Communication rate of data   |
| Data Bits                                | 8              | 5 to 8                         | Number of data bits used in the protocol (ASCII usually 7/RTU usually 8) |
| Stop Bits                                | 1              | 1 or 2                         | Number of stop bits used in the protocol                                 |
| Parity                                   | Even           | even, odd, none, space or mark | Parity of the protocol   |
| Station                                  | 0              | 0–99                           | Slave Number   |
| 0 – Signed Value ><br>1 – Unsigned Value | 0              | 0 or 1                         | 0: Signed values<br>1: Unsigned values                                   |
| Protocol                                 | RTU            | ASCII or RTU                   | Modbus protocol types  |
| Transaction Identifier                   | 0              | 0–1                            | Use or not use Transaction Identifier                                    |
| S – Serial or T – TCP/IP                 | —              | S or T                         | Choose S for Serial, or T for TCP/IP communication                       |

**Note:**  
 These parameters must be configured *exactly the same* as those you configured for the MODSL driver in the *Communications Parameters* dialog.

5. Click the **Advanced** button on the *Communication Parameters* dialog to open the *Advanced Settings* dialog:



**Advanced Settings Dialog**

6. Use the following table to specify the **Control RTS** (*Request to Send*) parameter:

| Parameter             | Default Value | Valid Values           | Description  |
|-----------------------|---------------|------------------------|--|
| Start message (ms)    | 1000          | 0 to 10000             | Maximum time to receive the beginning of the message from the device (time-out time).  |
| End message (ms)      | 0             | 0 to 10000             | Maximum time to receive the end of the message from the device from the beginning of the message. ( <b>Note:</b> The value zero means that the driver will not check these times.) |
| Interval between char | 500           | 0 to 10000             | Maximum time between the characters sent from the device.  |
| Wait CTS (ms)         | 100           | 0 to 10000             | Maximum time to receive the CTS signal after setting the RTS signal. ( <b>Note:</b> Valid only if the <b>Verify CTS</b> parameter has the <b>yes</b> value.)                       |
| Control RTS           | No            | No, Yes or yes+echo    | Define if the handshake signal of RTS (Request to Send) is set before communication and if there is an echo in the communication.  |
| Verify CTS            | No            | No or Yes              | Define if the driver must wait for the handshaking signal of CTS (Clear to Send) before sending a message.   |
| Disable DTR           | Not checked   | Not checked or checked | If checked, the driver will not set the DTR signal before starting the communication.  |
| Retries               | 0             | 0 to 9                 | The number of retries by each tag configured in the Driver worksheet if failure occurs.  |
| Tx Buffer (bytes)     | 512           | 0 to 512               | Maximum size of the buffer of information to be sent from the driver.  |
| Rx Buffer (bytes)     | 512           | 0 to 512               | Maximum size of the buffer of information to be received from the device.  |

**Notes:**

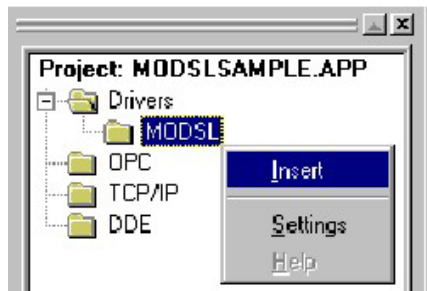
- Do not change any of the other default Advanced settings parameters at this time. You can consult the Studio *Technical Reference Manual* for information about configuring these parameters for future reference.
- Generally, you must change these parameters only if you are using a DCE (Data Communication Equipment) converter (such as 232/485), modem, and so forth between your computer, the driver, and the host. However, before adjusting the Advanced communication parameters, you must be familiar with the characteristics of the DCE.

### **Configuring the Standard Driver Worksheet**

This section explains how to configure a *Standard Driver Worksheet* (or Communication table) to associate application tags with the PLC addresses. You can configure multiple *Driver* worksheets — each of which is divided into a *Header* section and *Body* section.

Use the following steps to create a new Standard Driver Worksheet:

1. From the Studio development environment, select the **Comm** tab, located below the *Workspace* pane.
2. In the *Workspace* pane, expand the *Drivers* folder and right-click the *MODSL* subfolder.
3. When the pop-up menu displays, select the **Insert** option:

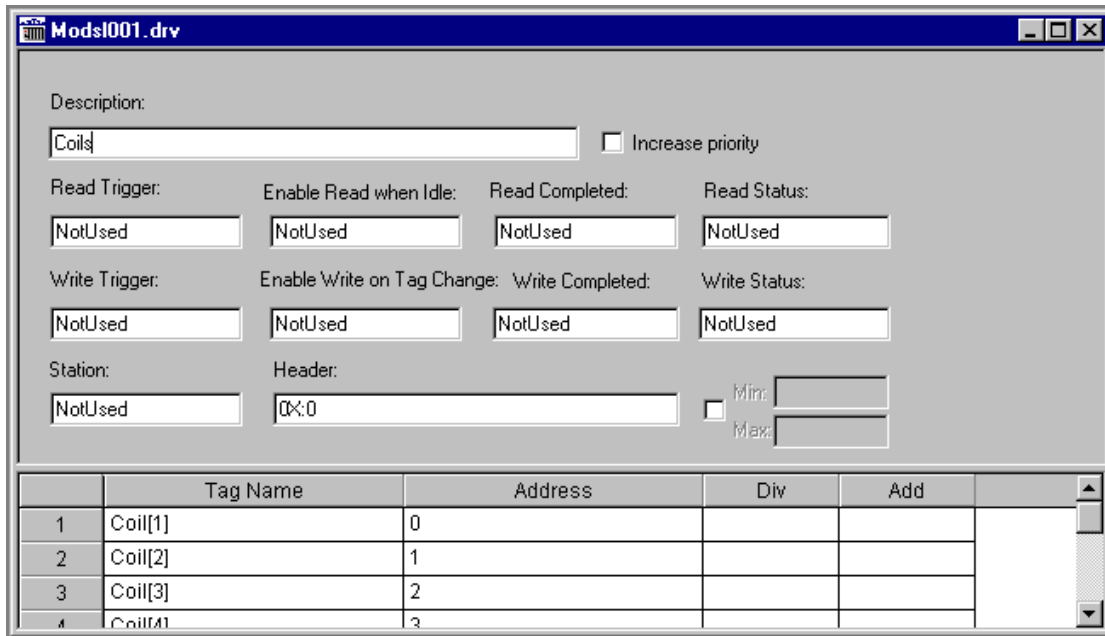


***Inserting a New Worksheet***

**Note:**

To optimize communication and ensure better system performance, you must tie the tags in different driver worksheets to the events that trigger communication between each tag group and the period in which each tag group must be read or written. Also, we recommend configuring the communication addresses in sequential blocks to improve performance.

The <drivername>.drv dialog displays (similar to the following figure):



**MODSL Driver Worksheet**

4. Use the following information to complete the **Station**, **Header**, and **Address** fields on this worksheet:
  - **Station** field: Use this field to specify the ID (*node*) of the device (*unit name*). Valid values are 0–247 (*no default*).
  - **Header** field: Use the information in the following table to define the type of variables that will be read from or written to the device and a reference to the initial address. (Default value is 0x: 0.)

These variables must comply with the following syntax:

**<Type>: <AddressReference>** (for example: **4X: 10**)

Where:

- \* **Type** is the register type. Type one of the following: **0X**, **1X**, **3X**, **4X**, **FP**, or **DW**.
- \* **AddressReference** is the initial address (reference) of the register type you configured.

After you edit the **Header** parameter, the system checks that the syntax is valid. If the syntax is invalid, Studio automatically inserts the default value (0x: 0) into the **Header** field.

Also, you can type a tag string in brackets {**Tag**} into the **Header** field, but you must be certain that the tag's value is correct and that you are using the correct syntax, or you will get an **Invalid Header** error.

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

| Header Field Information |               |                                  |  |
|--------------------------|---------------|----------------------------------|--|
| 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.   |
| 1                        | 1X:0          | Varies according to equipment    | Input status: Reads events using Modbus instruction 02.  |
| 3                        | 3X:0          | Varies according to equipment    | Input register: Reads events using Modbus instruction 04.  |
| 4                        | 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.                   |

- **Address** field: The body of the *Driver* worksheet allows you to associate each tag to its respective address in the device. Type the tag from your application database into the **Tag Name** column. This tag will receive values from or send values to an address on the device. The address must comply with the following syntax:

[**Signed/Unsigned**] <AddressOffset> (for example, 10) or  
 <AddressOffset>.<Bit> (for example: 10.3)

Where:

- \* **Signed/Unsigned** (*optional parameter used for integer values only*) If you do not specify this parameter, Studio inserts an integer value based on the parameters you set in the *Communication Parameters* dialog. Valid values are **S** (signed) and **U** (unsigned).
- \* **AddressOffset** is a parameter added to the **AddressReference** parameter (configured in the **Header** field) to compose the address to be read from or written to the device.
- \* **Bit** is the bit number (from 0–15) of the word address. This parameter is *optional* and can be combined with 3X or 4X **Header** configuration.

**➡ Cautions:**

- Studio stores the floating-point value in two, consecutive words. In this case, the **Address** value corresponds to the first Holding Register position. You must not configure a non-existing address or it will create a conflict.
- In the *Driver* worksheet, the start address is defined by the sum of the <AddressReference> parameter (from the **Header** field) and the lowest <AddressOffset> parameter (in the **Address** field). The resulting start address *must be a non-zero value*. A zero value will cause a conflict.
- You must not configure a range of addresses greater than the maximum block size (*data buffer length*) supported by each PLC within the same *Driver* worksheet. The maximum data buffer length for this driver is 64 bytes per *Standard Driver Worksheet*.

| Address Configuration Sample |              |               |
|------------------------------|--------------|---------------|
| Device Address               | Header Field | Address Field |
| 000001                       | 0x:0         | 0             |
| 000010                       | 0x:0         | 10            |
| 001020                       | 0x:1000      | 20            |
| 100001                       | 1x:0         | 1             |
| 100010                       | 1x:0         | 10            |
| 101020                       | 1x:1000      | 20            |
| 300001                       | 3x:0         | 1             |
| 300010                       | 3x:0         | 10            |
| 301020                       | 3x:1000      | 20            |
| 400001                       | 4x:0         | 1             |
| 400010                       | 4x:0         | 10            |
| 401020                       | 4x:1000      | 20            |
| 400001 and 400002            | FP:0         | 1             |
| 400013 and 400014            | FP:0         | 13            |
| 401022 and 401021            | DW:1000      | 21            |

DW and FP are special types. Sheets configured using these types must have all their addresses either Odd or Even. The following are examples that show how to configure these variables. The following example shows a wrong configuration.

**Example 1 – Floating point odd (401001–401002/401003–401004...401019–401020:**

The screenshot shows a configuration window titled "Modsl003.drv". It contains several input fields for "Description", "Read Trigger", "Write Trigger", "Station", and "Header". The "Header" field is set to "FP:1000". Below these fields is a table with 11 rows and 5 columns: "Tag Name", "Address", "Div", and "Add". The "Tag Name" column contains "FLOAT[1]" through "FLOAT[10]", and the "Address" column contains odd numbers from 1 to 19. The "Div" and "Add" columns are empty.

|    | 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 – Floating point even (401003–401004/401004–401005...401020–401021:**

The screenshot shows the configuration window for 'Modsl003.drv'. The description is 'FLOAT (1002-1020)'. The 'Increase priority' checkbox is unchecked. The 'Read Trigger' and 'Write Trigger' fields are empty. The 'Station' field is empty, and the 'Header' field contains 'FP:1000'. The 'Min.' and 'Max.' checkboxes are unchecked. Below the configuration fields is a table with 11 rows and 5 columns: Tag Name, Address, Div, and Add.

|    | 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 – Floating point wrong:**

The screenshot shows the configuration window for 'Modsl003.drv'. The description is 'FLOAT WRONG'. The 'Increase priority' checkbox is unchecked. The 'Read Trigger' and 'Write Trigger' fields are empty. The 'Station' field is empty, and the 'Header' field contains 'FP:1000'. The 'Min.' and 'Max.' checkboxes are unchecked. Below the configuration fields is a table with 11 rows and 5 columns: Tag Name, Address, Div, and Add.

|    | 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 |           |         |     |     |

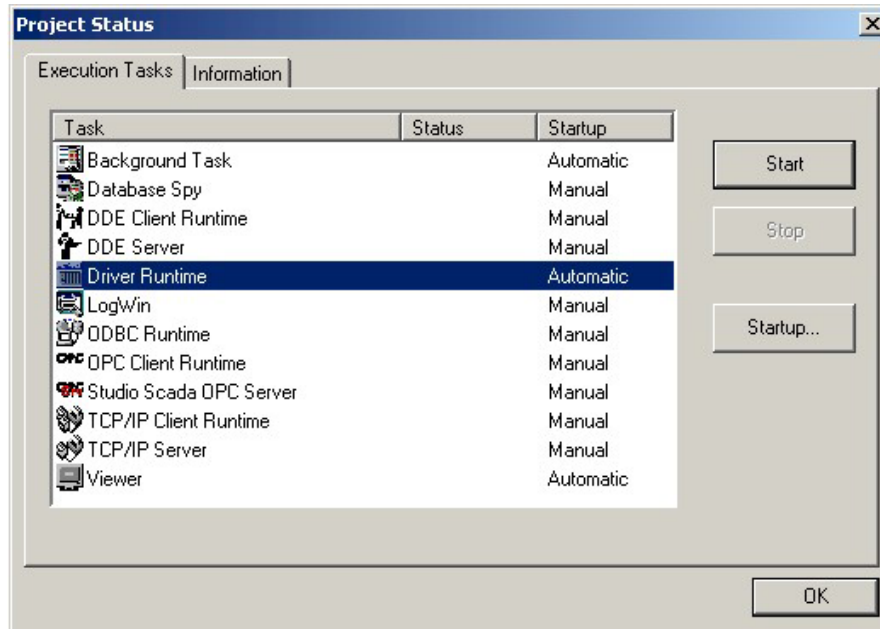
## Executing the Driver

After adding the MODSL driver to a project, Studio sets the project to execute the driver automatically when you start the run-time environment.

To verify that the driver run-time task is enabled and will start correctly, perform the following steps:

1. Select **Project** → **Status** from the main menu bar.

The *Project Status* dialog displays:



*Project Status Dialog*

2. Verify that the *Driver Runtime* task is set to **Automatic**.
  - If the setting is correct, click **OK** to close the dialog.
  - If the **Driver Runtime** task is set to **Manual**, select the **Driver Runtime** line. When the **Startup** button becomes active, click the button to toggle the *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 device, the tag you configured for the **Read Status** or **Write Status** fields 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 “Link Characteristics” for valid configuration). |
| 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 verify communication status using the Studio development environment *Output* window (*LogWin* module). To establish an event log for **Field Read Commands**, **Field Write Commands**, and **Serial Communication** right-click in the *Output* window. When the pop-up menu displays, select the option to set the log events. If you are testing a Windows CE target, you can enable the log at the unit (**Tools** → **LogWin**) and verify the `ce1log.txt` file created at the target unit.

If you are unable to establish communication with the PLC, try to establish communication between the PLC Programming Tool and the PLC. Quite frequently, communication is not possible because you have a hardware or cable problem, or a PLC configuration error. After successfully establishing communication between the device’s Programming Tool and the PLC, you can retest the supervisory driver.

To test communication with Studio, we recommend using the sample application provided rather than your new application.

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

You will find a sample application in the `/COMMUNICATION EXAMPLES/MODSL` directory. We strongly recommend that you use this sample application to test the MODSL driver before configuring your own customized application, for the following reasons:

- To better understand the information provided in the section of this document.
- To verify that your configuration is working satisfactorily.
- To certify that the hardware used in the test (device, adapter, cable, and PC) is working satisfactorily before you start configuring your own, customized applications.

 **Note:**

This application sample is not available for all drivers.

Use the following procedure to perform the test:

1. Configure the device's communication parameters using the manufacturer's documentation.
2. Open and execute the sample application.
3. Execute the *Viewer* module in Studio to display information about the driver communication.

 **Tip:**

You can use the sample application screen as the maintenance screen for your custom applications.

## 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.              |