Driver Manual

FS-8700-01 Modbus RTU & FS-8700-08 Modbus ASCII

APPLICABILITY & EFFECTIVITY
Effective for all systems manufactured after February 2021.

There are several similarities between these two drivers. They have been incorporated a single manual to ensure that information stays current. Although both drivers are referenced in this manual, they are different drivers and must be ordered separately.

Driver Revision: 4.12 & 1.03
Document Revision: 4.A

MSA Safety
1991 Tarob Court
Milpitas, CA 95035
Website: www.MSAsafety.com

U.S. Support Information:
+1 408 964-4443
+1 800 727-4377
Email: smc-support@msasafety.com

EMEA Support Information:
+31 33 808 0590
Email: smc-support.emea@msasafety.com

Contents

1 Modbus RTU/Modbus ASCII Description

The Modbus RTU and Modbus ASCII drivers allow the FieldServer to transfer data to and from devices using Modbus RTU or Modbus ASCII protocol respectively. Data can be transferred over either RS-232 or RS-485. The driver was developed for Modbus Application Protocol Specification V1.1a from Modbus-IDA. The specification can be found at www.modbus.org. The FieldServer can emulate either a Server or Client.

The information that follows describes how to expand upon the factory defaults provided in the configuration files included with the FieldServer.

There are various register mapping models being followed by various vendors.

To cover all these mappings, FieldServer uses the following three Models:

- **Modicon_5digit** – Use this format where addresses are defined in 0xxxx, 1xxxx, 3xxxx or 4xxxx format. A maximum of 9999 registers can be mapped of each type. This is FieldServer driver's default format.
- **ADU** – Application Data Unit address. Use this format where addresses of each type are defined in the range 1-65536.
- **PDU** – Protocol Data unit address. Use this format where addresses of each type are defined in the range 0-65535.

An example of the key difference between ADU and PDU:

- If Address_Type is ADU and address is 1, the driver will poll for register 0.
- If Address_Type is PDU, the driver will poll for address 1.

NOTE: If vendor document shows addresses in extended Modicon (i.e. 6 digit) format like 4xxxxx then consider these addresses as xxxxx (omit the first digit) and use either ADU or PDU.

NOTE: The purpose of providing 3 different ways of addressing the Modbus registers is to allow the user to choose the addressing system most compatible with the address list being used. At the protocol level, the same protocol specification is used for all three with the exception of the limited address range for Modicon_5digit.

2 Driver Scope of Supply

2.1 Supplied by MSA Safety for this Driver

Part #	Description
FS-8915-10	UTP cable (7 foot) for RS-232 use[1]

2.2 Provided by the Supplier of 3rd Party Equipment

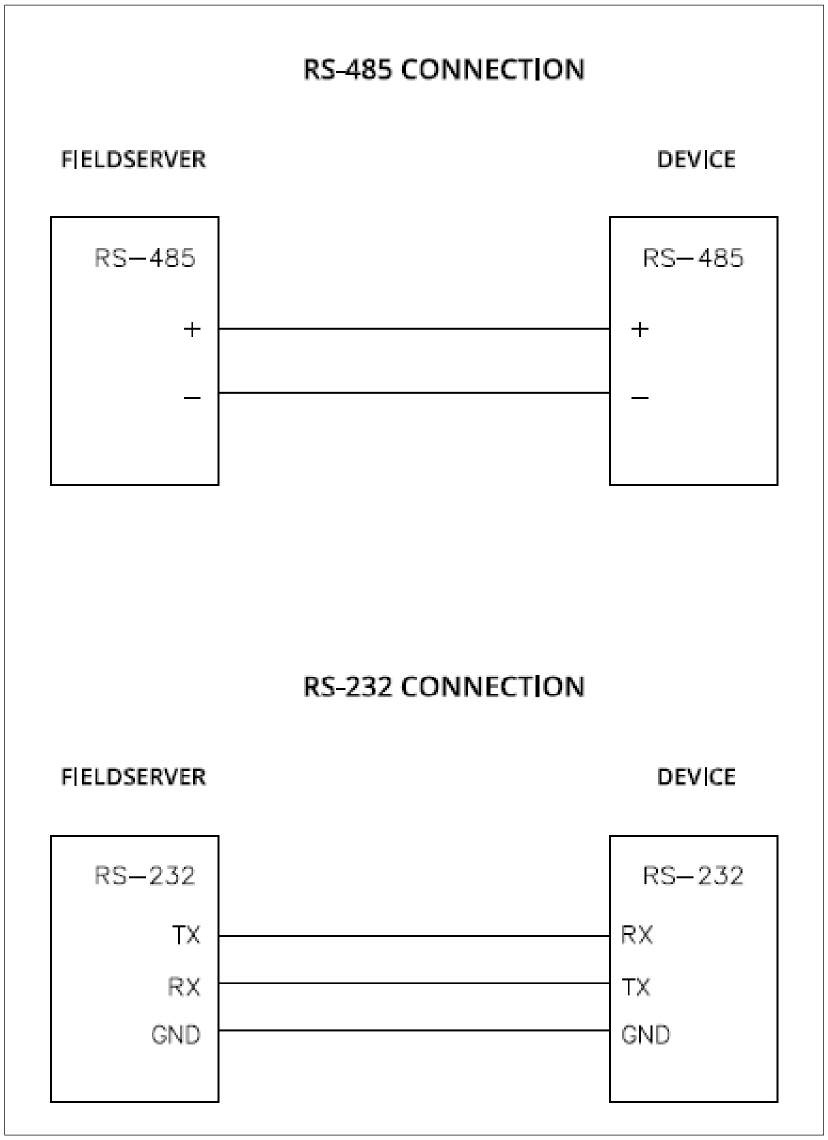
Part #	Description
	Modbus TRU or Modbus ASCII device

[1] This cable is necessary for connection to the driver. It is shipped with the FieldServer and not separately with the driver.

3 Hardware Connections

It is possible to connect a Modbus RTU or Modbus ASCII device to any of the existing serial ports on the FieldServer[2]. These ports simply need to be configured for the appropriate driver in the configuration file.

Configure the Modbus RTU or Modbus ASCII device according to manufacturer's instructions.



[2] Not all ports shown are necessarily supported by the hardware. Consult the appropriate Instruction manual for details of the ports available on specific hardware.

4 Data Array Parameters

Data Arrays are “protocol neutral” data buffers for storage of data to be passed between protocols. It is necessary to declare the data format of each of the Data Arrays to facilitate correct storage of the relevant data.

Section Title		
Data_Arrays		
Column Title	**Function**	**Legal Values**
Data_Array_Name	Provide name for Data Array.	Up to 15 alphanumeric characters
Data_Array_Format	Provide data format. Each Data Array can only take on one format.	Float, Bit, Byte, Uint16, Uint32, Sint16, Sint32
Data_Array_Length	Number of Data Objects. Must be larger than the data storage area required by the Map Descriptors for the data being placed in this array.	1-10000

Example

```
//    Data Arrays
Data_Arrays
Data_Array_Name     , Data_Array_Format     , Data_Array_Length
DA_AI_01            , UInt16                , 200
DA_AO_01            , UInt16                , 200
DA_DI_01            , Bit                   , 200
DA_DO_01            , Bit                   , 200
```

5 Configuring the FieldServer as a Modbus RTU or Modbus ASCII Client

For detailed information on FieldServer configuration, refer to the FieldServer Configuration Manual. The information that follows describes how to expand upon the factory defaults provided in the configuration files included with the FieldServer (see ".csv" sample files provided with the FieldServer).

This section documents and describes the parameters necessary for configuring the FieldServer to communicate with a Modbus RTU or Modbus ASCII Server.

NOTE: In the following tables, * indicates an optional parameter with the **bold** legal value as the default.

5.1 Client Side Connection Parameters

Section Title
Connections

Column Title	Function	Legal Values
Port	Specify which port the device is connected to the FieldServer.	P1-P2, R1-R2[3]
Baud*	Specify baud rate.	110 – 115200, **9600**; standard baud rates only
Parity*	Specify parity.	Even, Odd, **None**
Data_Bits*	Specify data bits.	7, **8**
Stop_Bits*	Specify stop bits.	**1**,2
Protocol	Specify protocol used.	**For Modbus RTU:** Modbus _RTU **For Modbus ASCII:** MB_ASCII
Poll_Delay*	Time between internal polls.	0-32000s, **0.05s**

Example:

```
//    Client Side Connections

Connections
Port    , Baud     , Parity     , Data_Bits     , Stop_Bits     , Protocol               , Poll_Delay
P1      , 9600     , None       , 8             , 1             , Modbus _RTU[4]         , 0.100s
```

[3] Not all ports shown are necessarily supported by the hardware. Consult the appropriate Instruction manual for details of the ports available on specific hardware.
[4] Change protocol to MB_ASCII to use Modbus ASCII protocol.

5.2 Client Side Node Parameters

Section Title		
Nodes		
Column Title	**Function**	**Legal Values**
Node_Name	Provide name for Node.	Up to 32 alphanumeric characters
Node_ID	Modbus station address of physical Server Node. Node_ID value 0 is reserved for broadcast messages.	0-255
Protocol	Specify protocol used.	**For Modbus RTU:** Modbus _RTU **For Modbus ASCII:** MB_ASCII
Port	Specify which port the device is connected to the FieldServer.	P1-P2, R1-R2[3]
Address_Type[5]	Specify Register Mapping Model.	ADU, PDU, -, **Modicon_5digit**
Write_Fnc*	Set to Multiple if Remote Server Node only supports Write Multiple function code 15 & 16.	Multiple, **-**
Write_Length*	Set to MD_Length if write-thru operation should write all registers as specified by MD_Length. By default, write-thru writes a single register. If Write_Length also specified on Map Descriptor, Map Descriptor's parameter will be used.	MD_Length, **-**, **1**

Example

```
//    Client Side Nodes
// For devices where 65536 addresses are available in each memory area
Nodes
Node_Name           , Node_ID       , Protocol            , Port       , Address_Type
Modbus device 1     , 1             , Modbus _RTU[6]      , P1         , ADU
Modbus device 2     , 2             , Modbus_RTU          , P1         , PDU

// For devices where only 9999 registers are available in each memory area
Nodes
Node_Name           , Node_ID       , Protocol            , Port
Modbus device 3     , 3             , Modbus_RTU          , P1
```

[5] Optional for Modicon 5 digit devices.
[6] Change protocol to MB_ASCII to use Modbus ASCII protocol.

5.3 Client Side Map Descriptor Parameters

5.3.1 FieldServer Specific Map Descriptor Parameters

Column Title	Function	Legal Values
Map_Descriptor_Name	Name of this Map Descriptor.	Up to 32 alphanumeric characters
Data_Array_Name	Name of Data Array where data is to be stored in the FieldServer.	One of the Data Array names from **Section 4**
Data_Array_Offset	Starting location in Data Array.	0 to (Data_Array_Length -1) as specified in **Section 4**
Function	Function of Client Map Descriptor.	WRBC, WRBX, RDBC, ARCS; refer to **Section 5.4** for more information

5.3.2 Driver Related Map Descriptor Parameters

Column Title	Function	Legal Values
Node_Name	Name of Node to fetch data from.	One of the Node names specified in **Section 5.2**
Data_Type[7]	Specify memory area. Refer to **Section 7.1.2** on how to transfer 32 Bit values using Modbus registers.	Address_Type = ADU
		Coil, Discrete_Input, Input_Register, Holding_Register, Single_Coil, Single_Register, Slave_ID (Section
		Address_Type = PDU
		FC01, FC02, FC03, FC04, FC05, FC06, FC15, FC16
		Address_Type = Modicon_5digit
		- (Dash), Single_Register, Single_Coil
		All Address_Type
		Float_Reg, 32Bit_Reg, Input_Float, Input_Reg_32Bit, Float_Reg_Swap, 32Bit_Reg_Swap, Input_Float_Swap, Input_Reg_32Bit_Swap (**Section 7.4.1**); Input_Reg_64Bit, 64Bit_Reg, Input_Double, Double_Reg (**Section 7.4.2**); Reg_Bytes, Input_Reg_Bytes, Reg_Bits, Input_Reg_Bits (**Section 7.3**); Device_ID (**Section 7.5.1**)
Address	Starting address of read block.	Address_Type = ADU
		1-65536
		Address_Type = PDU
		0-65535
		Address_Type = Modicon_5digit
		40001, 30001, etc.

[7] Optional only for Modicon_5digit addressing, and only if Single writes do not need to be forced.

Length*	Length of Map Descriptor.	**1**-125 (For Analog polls), **1**-800 (For Binary polls)
Scattered_Addresses*	Specify additional addresses to read on this map descriptor.	List of addresses separated by a space, all within quotation marks (**Section** 7.7)
Write_Length*	Set to MD_Length if write-thru operation should write all registers as specified by length parameter. By default, write-thru writes a single register.	MD_Length, -, **1**
Data_Array_Low_Scale*	Scaling zero in Data Array.	Any signed 32 bit integer in the range: -2147483648 to 2147483647, **0**
Data_Array_High_Scale*	Scaling max in Data Array.	Any signed 32 bit integer in the range: -2147483648 to 2147483647, **100**
Node_Low_Scale*	Scaling zero in Connected Node.	Any signed 32 bit integer in the range: -2147483648 to 2147483647, **0**
Node_High_Scale*	Scaling max in Connected Node.	Any signed 32 bit integer in the range: -2147483648 to 2147483647, **100**
Object_ID*	Only used with Data_Type Device_ID.	0 - 6 (**Section** 7.5.1)

5.3.3 Timing Parameters

Column Title	Function	Legal Values
Scan_Interval*	Rate at which data is polled.	0-32000, **1**

5.4 Map Descriptor Examples

NOTE: All three examples below are addressing the same Modbus registers.

```
//   Client Side Map Descriptors

// For Nodes where Address_Type is ADU
Map_Descriptors
Map_Descriptor_Name      , Data_Array_Name     , Data_Array_Offset    , Function    , Node_Name
CMD_AI_01                , DA_AI_01            , 0                    , Rdbc        , MODBUSDEVICE1
CMD_AO_01                , DA_AO_01            , 0                    , Rdbc        , MODBUSDEVICE1
CMD_DI_01                , DA_DI_01            , 0                    , Rdbc        , MODBUSDEVICE1
CMD_DO_01                , DA_DO_01            , 0                    , Rdbc        , MODBUSDEVICE1
```

```
, Data_Type              , Address      , Length      , Scan_Interval
, Input_Register         , 1            , 20          , 1.000s
, Holding_Register       , 1            , 20          , 1.000s
, Discrete_Input         , 1            , 20          , 1.000s
, Coil                   , 1            , 20          , 1.000s
```

```
// For Nodes where Address_Type is PDU
Map_Descriptors
Map_Descriptor_Name  , Data_Array_Name  , Data_Array_Offset  , Function  , Node_Name,        , Data_Type  , Address  , Length  ,Scan_Interval
CMD_AI_02            , DA_AI_02         , 0                  , Rdbc      , MODBUS DEVICE2   , FC04       , 0        , 20      , 1.000s
CMD_AO_02            , DA_AO_02         , 0                  , Rdbc      , MODBUS DEVICE2   , FC03       , 0        , 20      , 1.000s
CMD_DI_02            , DA_DI_02         , 0                  , Rdbc      , MODBUS DEVICE2   , FC02       , 0        , 20      , 1.000s
CMD_DO_02            , DA_DO_02         , 0                  , Rdbc      , MODBUS DEVICE2   , FC01       , 0        , 20      , 1.000s
```

```
// For Nodes where Address_Type is Modicon_5digit
Map_Descriptors
Map_Descriptor_Name  , Data_Array_Name  , Data_Array_Offset  , Function  , Node_Name,        , Address  , Length  , Scan_Interval
CMD_AI_03            , DA_AI_03         , 0                  , Rdbc      , MODBUS DEVICE3   , 30001    , 20      , 1.000s
CMD_AO_03            , DA_AO_03         , 0                  , Rdbc      , MODBUS DEVICE3   , 40001    , 20      , 1.000s
CMD_DI_03            , DA_DI_03         , 0                  , Rdbc      , MODBUS DEVICE3   , 10001    , 20      , 1.000s
CMD_DO_03            , DA_DO_03         , 0                  , Rdbc      , MODBUS DEVICE3   , 00001    , 20      , 1.000s
```

6 Configuring the FieldServer as a Modbus RTU or Modbus ASCII Server

For detailed information on FieldServer configuration, refer to the FieldServer Configuration Manual. The information that follows describes how to expand upon the factory defaults provided in the configuration files included with the FieldServer (see ".csv" sample files provided with the FieldServer).

This section documents and describes the parameters necessary for configuring the FieldServer to communicate with a Modbus RTU or Modbus ASCII Client.

The configuration file tells the FieldServer about its interfaces, and the routing of data required. In order to enable the FieldServer for Modbus RTU or Modbus ASCII communications, the driver independent FieldServer buffers need to be declared in the "Data Arrays" section, the FieldServer virtual Node(s) needs to be declared in the "Server Side Nodes" section, and the data to be provided to the clients' needs to be mapped in the "Server Side Map Descriptors" section. Details on how to do this can be found below.

NOTE: In the following tables, * indicates an optional parameter with the **bold** legal value as the default.

6.1 Server Side Connection Parameters

Section Title		
Connections		
Column Title	**Function**	**Legal Values**
Port	Specify which port the device is connected to the FieldServer.	P1-P2, R1-R2[8]
Baud*	Specify baud rate.	110 – 115200, **9600**; standard baud rates only
Parity*	Specify parity.	Even, Odd, **None**
Data_Bits*	Specify data bits.	7, **8**
Stop_Bits*	Specify stop bits.	**1**, 2
Protocol	Specify protocol used.	**For Modbus RTU:** Modbus _RTU **For Modbus ASCII:** MB_ASCII
Framing_Timeout*	Sets the time to wait for a message frame to complete on the network. This is useful on busy Modbus networks where unknown messages for other devices may cause longer timeouts.	**0** - 2147483647 milliseconds (0 means disabled)
Accept_Broadcast*	Specify server to accept broadcast messages.	Yes, **No**

Example

Change protocol to MB_ASCII to use Modbus ASCII protocol.

```
//    Server Side Connections

Connections
Port            , Baud    , Parity    , Data_Bits    , Stop_Bits    , Protocol
P1              , 9600    , None      , 8            , 1            , Modbus_RTU
```

[8] Not all ports shown are necessarily supported by the hardware. Consult the appropriate Instruction manual for details of the ports available on specific hardware.

6.2 Server Side Node Parameters

Section Title		
Nodes		
Column Title	**Function**	**Legal Values**
Node_Name	Provide name for Node.	Up to 32 alphanumeric characters
Node_ID	Node ID of physical Server Node.	1 – 255
Protocol	Specify protocol used.	Modbus RTU
Address_Type*[9]	Specify Register Mapping Model.	ADU, PDU, **-**, **Modicon_5digit**
Node_Offline_Response*	Set the FieldServer response to the Modbus RTU Client when the Server Node supplying the data has gone offline.	**Gateway_Device_Failed**, **Exception_B**, No_Response, Old_Data, Zero_Data, FFFF_Data; See **Section 7.2**
Node_Description*	Specify Node description text.	Any string up to 99 characters long, **-**
Partial_Data_Response*	Set the FieldServer's response to the Modbus RTU Client request when addresses are not defined in the Map Descriptor section.	**Do_not_Respond**, Fill_Gaps_With_Zero, Fill_Gaps_With_FFFF

Example

Change protocol to MB_ASCII to use Modbus ASCII protocol.

```
//    Server Side Nodes
// For devices where 65536 addresses are available in each memory area
Nodes
Node_Name          , Node_ID          , Protocol                , Address_Type
MB_Srv_11          , 11               , Modbus_RTU              , ADU
MB_Srv_12          , 12               , Modbus_RTU              , PDU

// For devices where only 9999 registers are available in each memory area
Nodes
MB_Srv_13          , 13               , Modbus_RTU              , Modicon_5digit
MB_Srv_14          , 14               , Modbus_RTU              , -
```

6.3 Server Side Map Descriptor Parameters

6.3.1 FieldServer Specific Map Descriptor Parameters

Column Title	Function	Legal Values
Map_Descriptor_Name	Name of this Map Descriptor.	Up to 32 alphanumeric characters
Data_Array_Name	Name of Data Array where data is to be stored in the FieldServer.	One of the Data Array names from **Section 4**
Data_Array_Offset	Starting location in Data Array.	0 to (Data_Array_Length -1) as specified in **Section 4**
Function	Function of Server Map Descriptor.	Passive, server

[9] Optional for Modicon 5 digit devices.

6.3.2 Driver Related Map Descriptor Parameters

Column Title	Function	Legal Values
Node_Name	The name of the Node being represented.	One of the Node names specified in **Section 6.2**
Data_Type[10]	Specify memory area.	Address_Type = ADU
		Coil, Discrete_Input, Input_Register, Holding_Register, Single_Coil, Single_Register, Slave_ID (**Section 6.4.1**)
		Address_Type = PDU
		FC01, FC02, FC03, FC04, FC05, FC06, FC15, FC16
		Address_Type = Modicon_5digit
		- (Dash), Single_Register, Single_Coil
		All Address_Type
		Device_ID (**7.5.2**)
Address	Starting address of read block.	Address_Type = ADU
		1-65536
		Address_Type = PDU
		0-65535
		Address_Type = Modicon_5digit
		40001, 30001, etc.
Length*	Length of Map Descriptor.	Address_Type = ADU
		1-65536
		Address_Type = PDU
		1-65536
		Address_Type = Modicon_5digit
		1-9999
Data_Array_Low_Scale*	Scaling zero in Data Array.	Any signed 32 bit integer in the range: -2147483648 to 2147483647, **0**
Data_Array_High_Scale*	Scaling max in Data Array.	Any signed 32 bit integer in the range: -2147483648 to 2147483647, **100**
Node_Low_Scale*	Scaling zero in Connected Node.	Any signed 32 bit integer in the range: -2147483648 to 2147483647, **0**
Node_High_Scale*	Scaling max in Connected Node.	Any signed 32 bit integer in the range: -2147483648 to 2147483647, **100**

[10] Optional only for Modicon_5digit addressing, and only if Single writes do not need to be forced.

6.4 Map Descriptor Examples

NOTE: All three examples below are addressing the same Modbus registers.

```
//   Server Side Map Descriptors where Node Address_Type is ADU
Map_Descriptors
Map_Descriptor_Name    , Data_Array_Name    , Data_Array_Offset    , Function    , Node_Name    , Data_Type
SMD_AI_01              , DA_AI_01           , 0                    , Passive     , MB_Srv_11    , Input_Register
SMD_AO_01              , DA_AO_01           , 0                    , Passive     , MB_srv_11    , Holding_Register
```

```
, Address    , Length    , Data_Array_Low_Scale    , Data_Array_High_Scale    , Node_Low_Scale    , Node_High_Scale
, 1          , 200       , 0                       , 100                      , 0                 , 10000
, 1          , 200       , 0                       , 100                      , 0                 , 10000
```

```
//   Server Side Map Descriptors where Node Address_Type is PDU
Map_Descriptors
Map_Descriptor_Name    , Data_Array_Name    , Data_Array_Offset    , Function    , Node_Name    , Data_Type
SMD_AI_02              , DA_AI_02           , 0                    , Passive     , MB_Srv_12    , FC04
SMD_AO_02              , DA_AO_02           , 0                    , Passive     , MB_srv_12    , FC03
```

```
, Address    , Length    , Data_Array_Low_Scale    , Data_Array_High_Scale    , Node_Low_Scale    , Node_High_Scale
, 0          , 200       , 0                       , 100                      , 0                 , 10000
, 0          , 200       , 0                       , 100                      , 0                 , 10000
```

```
// For Nodes where Address_Type is Modicon_5digit
Map_Descriptors
Map_Descriptor_Name    , Data_Array_Name    , Data_Array_Offset    , Function    , Node_Name
SMD_AI_01              , DA_AI_01           , 0                    , Passive     , MBP_Srv_13
SMD_AO_01              , DA_AO_01           , 0                    , Passive     , MBP_Srv_13
```

```
, Address    , Length    , Data_Array_Low_Scale    , Data_Array_High_Scale    , Node_Low_Scale     Node_High_Scale
, 30001      , 200       , 0                       , 100                      , 0                 , 10000
, 40001      , 200       , 0                       , 100                      , 0                 , 10000
```

6.4.1 Slave_ID

The Node_Description will automatically be used to respond to the Report Slave_ID request (Function Code 17 or FC17). If the Node_Description is not defined the title in the common information section will be used as the description in the Slave_ID response.

6.4.2 Slave_ID Lookup in Table

The Slave_ID will be read from device PLC_21. The response will be searched for occurrences of the strings in the table in column table string. If a match is found the **user** value will be written.

The Table_String must occur in the Slave_ID in any position.

```
Config_Table
Config_Table_Name          , Table_String        , Table_Index_Value            , Table_User_Value
slave_id_profile           , FS01                , 1                            , 1
slave_id_profile           , FS02                , 2                            , 2
slave_id_profile           , FS03                , 3                            , 3
```

```
Map_Descriptors
Map_Descriptor_Name    , Data_Array_Name    , Data_Array_Offset    , Function    , Node_Name    , Scan_Interval
CMD_DO_01              , DA_DO_01           , 0                    , RDBC        , PLC_21       , 0.000s
```

```
, Data_Type    , Length    , Config_Table_Name
, Slave_Id     , 1         , slave_id_profile
```

7 Useful Features – Modbus RTU

7.1 Managing Floating Points with Modbus

Modbus as a standard does not support floating point formats. Many vendors have written higher level communications software to use two 16 bit registers to represent floating point or 32 bit integers. This requires conversion software on both ends of the communication channel. The FieldServer supports this function and also provides other options to resolve this issue.

7.1.1 Transferring Non-integer Values with One Register

It is possible to represent values higher than 32767 using one register in one of two ways:

- Declare data arrays as type Uint16 (Unsigned integer); this allows a range from 0 to 65535.
- Use the scaling function on the FieldServer, which allows any range with 16 bit resolution.

The following example shows how scaling can be achieved on the Server side of the configuration. Note that scaling can also be done on the Client side to scale down a value that was scaled up by a Modbus vendor. Further information regarding scaling can be found in the FieldServer Configuration manual.

Example

```
Map_Descriptors
Map_Descriptor_Name   , Data_Array_Name   , Data_Array_Offset   , Function   , Node_Name    , Address   , Length
SMD_AI1               , DA_AI_01          , 0                   , Passive    , MBP_Srv_11   , 30001     , 200
SMD_AO1               , DA_AO_01          , 0                   , Passive    , MBP_Srv_11   , 40001     , 200
```

```
, Data_Array_Low_Scale   , Data_Array_High_Scale   , Node_Low_Scale   , Node_High_Scale
, 0                      , 100                     , 0                , 10000
, 0                      , 100                     , 0                , 10000
```

This example multiplies the values in the data array by 100 (10000 on Node_High_Scale is 100X larger than 100 on Data_Array_High_Scale). This is most commonly used when the user wants to introduce values after the decimal point. For example, a value of 75.6 will be sent as 7560, which can then be rescaled by the Modbus master.

7.1.2 Transferring Float/32 bit Values with Two Registers

If a Modbus Server sends two consecutive registers to the FieldServer representing either a floating point value or a 32 bit integer value, the FieldServer can combine and decode these registers back into their original format. To do this, declare Data Array of type Float or UINT32 and set the Map Descriptor Data_Type as 'Float_Reg', '32Bit_Reg', etc.

Example

```
Data_Arrays
Data_Array_Name   , Data_Format   , Data_Array_Length
DA1               , Float         , 20
DA2               , UInt32        , 20
DA3               , Float         , 20
DA4               , UInt32        , 20
```

```
//   Client Side Map Descriptors where Nodes where Address_Type is PDU
Map_Descriptors
Map_Descriptor_Name   , Data_Array_Name   , Data_Array_Offset   , Function   , Node_Name,
CMD_AO_01             , DA1               , 0                   , Rdbc       , Modbus Device2
CMD_AO_02             , DA2               , 0                   , Rdbc       , Modbus Device2
CMD_AI_01             , DA3               , 0                   , Rdbc       , Modbus Device2
CMD_AI_02             , DA4               , 0                   , Rdbc       , Modbus Device2
```

```
, Data_Type         , Address   , Length   , Scan_Interval
, Float_Reg         , 0         , 20       ,1.000s
, 32Bit_Reg         , 0         , 20       ,1.000s
, Input_Float       , 0         , 20       ,1.000s
, Input_Reg_32Bit   , 0         , 20       ,1.000s
```

Each Map Descriptor will read 20 pairs of registers and store them as a 32-bit floating number or a 32-bit Integer.

If the server device sends swapped registers (low value register first) then use the corresponding _swap data_types.

NOTE: The value in the address parameter can be ADU, PDU or Modicon 5-digit types; see Section 5.3.2.

7.2 Node_Offline_Response

This function is specific to the Modbus RTU driver.

In systems where data is being collected from multiple Server Nodes and made available on a FieldServer configured as a Modbus RTU Server, when a Server Node goes offline the default behavior of the FieldServer would be to stop responding to polls for this data. This might not be what the user wants. Various options exist making it possible to signal that the data quality has gone bad without creating error conditions in systems sensitive to the default option.

The following options can be configured under the Node parameter, Node_Offline_Response, to set the response of the FieldServer to the Modbus RTU Client when the Server Node supplying the data is offline:

- **No_Response** - This is the default option. The FieldServer simply does not respond when the corresponding Server Node is offline.
- **Old_Data** - The FieldServer responds with the last known data value. This maintains the communication link in an active state but may hide the fact that the Server Node is offline.
- **Zero_Data** - The FieldServer responds with the data values set to zero. If the user expects non-zero values, this option will signal the offline condition without disrupting communications.
- **FFFF_Data** - The FieldServer responds with data values set to FFFF (hex). If the user expects other values, this option will signal the offline condition without disrupting communications.

When configured as a Server this parameter can force a desired exception response as follows:

- Node_Offline_Message or Exception_4 - FieldServer's response will be Exception 4.
- Gateway_Path_Unavailable or Exception_A - FieldServer's response will be Exception A.
- Gateway_Device_Failed or Exception_B - FieldServer's response will be Exception B.

<u>Example</u>

```
Nodes
Node_Name   , Node_ID   , Protocol      , Node_Offline_Response      , Port
DEV11       , 11        , Modbus_RTU    , No_Response                , -
DEV12       , 12        , Modbus_RTU    , Old_Data                   , -
DEV15       , 15        , Modbus_RTU    , Zero_Data                  , -
DEV16       , 16        , Modbus_RTU    , FFFF_Data                  , -
DEV17       , 17        , Modbus_RTU    , Exception_4                , -
DEV18       , 18        , Modbus_RTU    , Gateway_Path_Unavailable   , -
```

7.3 Splitting Registers into Bytes or Bits

Sometimes it is required to split a register into Bytes or bits. The following Map Descriptors read registers and store the bytes/bits in consecutive data array locations. The FieldServer will store the least significant byte/bit at the 1st offset and will continue sequentially. To implement this feature, declare a Data Array with Data_Format Byte or bit and use that Data_Array_Name when setting up the Map_Descriptor parameters. In the Map_Descriptors, set Data_Type as ‘Reg_Bytes’ or ‘Reg_Bits’ according to the Data_Format of the Data_Array.

Example

```
Data_Arrays
Data_Array_Name   , Data_Format   , Data_Array_Length
DA1               , Byte          , 40
DA2               , Byte          , 40
DA3               , Bit           , 320
DA4               , Bit           , 320
```

```
//Client Side Map Descriptors for Nodes where Address_Type is PDU

Map_Descriptors
Map_Descriptor_Name , Data_Array_Name , Data_Array_Offset , Function , Node_Name     , Data_Type       , Address , Length , Scan_Interval
CMD_AO_01           , DA1             , 0                 , RDBC     , ModbusDevice2 , Reg_Bytes       , 0       , 40     , 1.000s
CMD_AO_02           , DA2             , 0                 , RDBC     , ModbusDevice2 , Input_Reg_Bytes , 0       , 40     , 1.000s
CMD_AI_01           , DA3             , 0                 , RDBC     , ModbusDevice2 , Reg_Bytes       , 0       , 320    , 1.000s
CMD_AI_02           , DA4             , 0                 , RDBC     , ModbusDevice2 , Input_Reg_Bytes , 0       , 320    , 1.000s
```

Each Map Descriptor will read 20 registers and store them as 40 bytes or 320 bits.

7.4 Reading Data Types

7.4.1 32-Bit Integer and Float Data Types

When a Modbus slave device needs to pass a 32-Bit Integer or a 32-Bit Float, it splits the float into two 16-bit Integers and maps it to consecutive registers. The following data types read the 2 consecutive registers and auto combines them into a 32-Bit Integer or Float before storing it in a Data Array.

- Float_Reg (32-Bit IEEE 754 Floating Point in Holding Register FC03)
- 32Bit_Reg (32-Bit Integer in Holding Register FC03)
- Input_Float (32-Bit IEEE 754 Floating Point in Input Register FC04)
- Input_Reg_32Bit (32-Bit Integer in Input Register FC04)
- Float_Reg_Swap (32-Bit IEEE 754 Floating Point with swapped format in Holding Register FC03)
- 32Bit_Reg_Swap (32-Bit Integer with swapped format in Holding Register FC03)
- Input_Float_Swap (32-Bit IEEE 754 Floating Point with swapped format in Input Register FC04)
- Input_Reg_32Bit_Swap (32-Bit Integer with swapped format in Input Register FC04)

Example

```
Data_Arrays
Data_Array_Name   , Data_Format   , Data_Array_Length
DA_U32_01         , Uint32        , 20
DA_FLT_01         , Float         , 20
```

```
//Client Side Map Descriptors

Map_Descriptors
Map_Descriptor_Name ,                 ,                  ,          ,           , Data_Type            ,         ,        ,
                    Data_Array_Name   Data_Array_Offset  Function   Node_Name                          Address   Length   Scan_Interval
CMD_01              , DA_FLT_01       , 0                , RDBC     , Dev_01    , Float_Reg            , 40001   , 1      , 0.000s
CMD_02              , DA_U32_01       , 0                , RDBC     , Dev_01    , 32Bit_Reg            , 40003   , 1      , 0.000s
CMD_03              , DA_FLT_01       , 1                , RDBC     , Dev_01    , Input_Float          , 30001   , 1      , 0.000s
CMD_04              , DA_U32_01       , 1                , RDBC     , Dev_01    , Input_Reg_32Bit      , 30003   , 1      , 0.000s
CMD_05              , DA_FLT_01       , 2                , RDBC     , Dev_01    , Float_Reg_Swap       , 40005   , 1      , 0.000s
CMD_06              , DA_U32_01       , 2                , RDBC     , Dev_01    , 32Bit_Reg_Swap       , 40007   , 1      , 0.000s
CMD_07              , DA_FLT_01       , 3                , RDBC     , Dev_01    , Input_Float_Swap     , 30005   , 1      , 0.000s
CMD_08              , DA_U32_01       , 3                , RDBC     , Dev_01    , Input_Reg_32Bit_Swap , 30007   , 1      , 0.000s
```

7.4.2 64-Bit Integer and Float Data Types

When a Modbus slave device needs to pass a 64-Bit Integer or a 64-Bit Float, it splits the float into four 16-bit Integers and maps it to consecutive registers. The following data types read the 4 consecutive registers and auto combines them into a 64-Bit Integer or Float, before the scaling is applied (to keep the decimal precision) and stores the scaled value in a Data Array. When serving the value to the output protocol, the reverse scaling needs to be applied.

- 64Bit_Reg (64-Bit Integer in Holding Register FC03)
- Double_Reg (64-Bit IEEE 754 Floating Point in Holding Register FC03)
- Input_Reg_64bit (64-Bit Integer in Input Register FC04)
- Input_Double (64-Bit IEEE 754 Floating Point in Input Register FC04)

Example

```
Data_Arrays
Data_Array_Name   , Data_Format    , Data_Array_Length
DA_U32_01         , Uint32         , 20
DA_FLT_01         , Float          , 20
```

```
//Client Side Map Descriptors

Map_Descriptors
Map_Descriptor_Name  , Data_Array_Name  , Data_Array_Offset  , Function  , Node_Name  , Data_Type        , Address  , Length  , Scan_Interval
CMD_01               , DA_U32_01        , 0                  , RDBC      , Dev_01     , 64Bit_Reg        , 40001    , 1       , 0.000s
CMD_02               , DA_FLT_01        , 0                  , RDBC      , Dev_01     , Double_Reg       , 40003    , 1       , 0.000s
CMD_03               , DA_U32_01        , 1                  , RDBC      , Dev_01     , Input_Reg_64bit  , 30001    , 1       , 0.000s
CMD_04               , DA_FLT_01        , 1                  , RDBC      , Dev_01     , Input_Double     , 30003    , 1       , 0.000s
```

```
, Data_Array_Low_Scale  , Data_Array_High_Scale  , Node_Low_Scale  , Node_High_Scale
, 0                     , 1                      , 0               , 1000000000
, 0                     , 1                      , 0               , 1000000000
, 0                     , 1                      , 0               , 1000000000
, 0                     , 1                      , 0               , 1000000000
```

```
Nodes
Node_Name        , Node_ID    , Protocol
Virtual_Dev_11   , 11         , Bacnet_IP
```

```
//Client Side Map Descriptors

Map_Descriptors
Map_Descriptor_Name  , Data_Array_Name  , Data_Array_Offset  , Function  , Node_Name       , Object Type  , Object Instance
SMD_11_AI_01         , DA_U32_01        , 0                  , Passive   , Virtual_Dev_11  , AI           , 1
SMD_11_AI_02         , DA_FLT_01        , 0                  , Passive   , Virtual_Dev_11  , AI           , 2
SMD_11_AI_03         , DA_U32_01        , 1                  , Passive   , Virtual_Dev_11  , AI           , 3
SMD_11_AI_04         , DA_FLT_01        , 1                  , Passive   , Virtual_Dev_11  , AI           , 4
```

```
, Data_Array_Low_Scale  , Data_Array_High_Scale  , Node_Low_Scale  , Node_High_Scale
, 0                     , 1                      , 0               , 1000000000
, 0                     , 1                      , 0               , 1000000000
, 0                     , 1                      , 0               , 1000000000
, 0                     , 1                      , 0               , 1000000000
```

7.5 Reading Device Identification

7.5.1 Client Side Map Descriptor

There could be various objects describing device identification.

Each object has its own ID (0-255). Only the first 7 ID's (0-6) objects are defined and are ASCII Strings.

Object ID	Object Name
0	VendorName
1	ProductCode
2	MajorMinorRevision
3	VendorUrl
4	ProductName
5	ModelName
6	UserApplicationName

The Client Side Map Descriptors read the specified object from a remote device. They will store the object data character-by-character in the specified data array, up to the limit specified by the Map Descriptor length.

Any object could have up to 248 characters.

```
//Client Side Map Descriptors

Map_Descriptors
Map_Descriptor_Name   , Data_Array_Name   , Data_Array_Offset   , Function   , Node_Name   , Data_Type   , Address   , Length
CMD_Vendor_name       , DA_Vendor         , 0                   , Server     , PLC_01      , Device_Id   , 0         , 248
CMD_Product_code      , DA_Prodcode       , 0                   , Server     , PLC_01      , Device_Id   , 1         , 248
```

7.5.2 Server Side Map Descriptor

Server Side Map Descriptors can define any object as shown below:

The Driver will serve strings from the data array as an object value.

The string from the data array is considered complete if the character is 0 (null) or if all characters are fetched as per the Map Descriptor length.

If the first character is 0 (null) then the single character '-' will be used as the object value.

```
//Server Side Map Descriptors

Map_Descriptors
Map_Descriptor_Name   , Data_Array_Name   , Data_Array_Offset   , Function   , Node_Name   , Data_Type   , Address   , Length
CMD_Vendor_name       , DA_Vendor         , 0                   , Server     , PLC_01      , Device_Id   , 0         , 248
CMD_Product_code      , DA_Prodcode       , 0                   , Server     , PLC_01      , Device_Id   , 1         , 248
```

7.6 Broadcasting Write Messages

This function is specific to the Modbus RTU driver.

Standard Modbus RTU node addresses range from 1 to 254, with 0 being reserved for broadcast messages. Setting the Node ID to 0 allows write messages to be broadcast to all configured slave devices.

To perform a valid broadcast, the node ID will need to be set to 0 and the map-descriptor function will need to be set to a write.

Example

```
// Client Side Nodes
Nodes
Node_Name        , Node_ID   , Protocol      , Port   , Address_Type
BROADCAST_NODE   , 0         , Modbus_RTU    , R2     , PDU
```

```
//Client Side Map Descriptors
Map_Descriptors
Map_Descriptor_Name , Data_Array_Name , Data_Array_Offset , Function , Node_Name     , Data_Type       , Address , Length , Scan_Interval
CMD_AO_01           , DA1             , 0                 , RDBC     , ModbusDevice2 , Reg_Bytes       , 0       , 40     , 1.000s
CMD_AO_02           , DA2             , 0                 , RDBC     , ModbusDevice2 , Input_Reg_Bytes , 0       , 40     , 1.000s
```

7.7 Reading Scattered Addresses

This function enables the user to read non-contiguous registers. It also avoids multiple polls using function 3 or 4 to read non-contiguous registers.

The following is an example to show the request and response to read input registers (sub function 0x04) 101 (0x65), 114 (0x72) and 149 (0x95) from Slave_ID 33 (0x21).

Poll request example:

Parameter	Value
Modbus slave address	0x21
Function code	0x64
Data length in bytes	0x06
Sub-function code	0x04 (0x03 to read holding registers)
Transmission number	0xXX
Address of 1st word to read (MSB)	0x00
Address of 1st word to read (LSB)	0x65
Address of 2nd word to read (MSB)	0x00
Address of 2nd word to read (LSB)	0x72
Address of 3rd word to read (MSB)	0x00
Address of 3rd word to read (LSB)	0x95
CRC MSB	0xXX
CRC LSB	0xXX

Suppose the value of register 101 is 3000, register 114 is 6000 and register 149 is 9000. The following would be the response from the slave:

Parameter	Value
Modbus slave address	0x21
Function code	0x64
Data length in bytes	0x06
Sub-function code	0x04 (same as in request)
Transmission number	0xXX (same as in request)
1st register value (MSB)	0x0B
1st register value (LSB)	0xB8
2nd register value (MSB)	0x17
2nd register value (LSB)	0x70
3rd register value (MSB)	0x23
3rd register value (LSB)	0x28
CRC MSB	0xXX
CRC LSB	0xXX

Following is the corresponding Client Map Descriptor example, it will read 3 scattered addresses and will store in data array DA_AI_01 at offset 0, 1 and 2.

```
//Client Side Map Descriptors
Map_Descriptors
Map_Descriptor_Name   , Data_Array_Name   , Data_Array_Offset   , Function   , Node_Name   , Address   , Length   , Scattered_Addresses
CMD_Scattered_Read    , DA_AI_01          , 0                   , RDBC       , PLC_33      , 30102     , 3        , "30115 30150"
```

On the server side, no configuration changes are required to support the scattered read function. Ensure that all registers are configured. Registers can be configured in a single Server Map Descriptor range or scattered over multiple Server Map Descriptors.

8 Troubleshooting

8.1 Server Configuration of System Station Address

When using the FieldServer as a Modbus Server, the FieldServer System Station address must be configured to be different from any of the configured Modbus Server Node_ID's. Configuring these to be the same invokes proprietary system information to be transmitted and should therefore be avoided.

9 Vendor Information

9.1 Connection to York Modbus Microgateway

If connecting the FieldServer to a York Modbus Microgateway, the Node_ID of the Microgateway is defined by the address DIP switches. If switch 4 is set to 'On' and the other switches are set to 'off' then Node_ID of the Microgateway is '247', the parity is 'Even', and the stop bits are 1. Other Node_ID combinations can be found in the York Modbus Microgateway Installation Manual.

9.2 Modbus ASCII - Examples of FieldServer Setup for Typical Clients

9.2.1 FieldServer with GE Cimplicity as Client

- Run the Cimplicity "Workbench" and create a "New Project" with a unique "Project Name" option of "Basic Control" and protocol "Modbus ASCII".
- Check the project properties and continue with the "Project Wizard Setup" that appears.
- Add Modbus port giving it a description.
- Create and configure the devices, select "new item".
- Name the device, select the port, give it a description (e.g. FieldServer), and choose "SYSTEM" resource.
- Create and configure the points.
- Select "new item", name the point and choose the appropriate device.
- Under the "General" tab, point properties require a description.

NOTE: The elements must have a value greater than 8.

- Under the "Device" tab, properties need the appropriate address (e.g. 40001 and also require the leading 0's), change the update criteria to "On Scan".
- When the project is configured, run by pressing the "play" button.
- Expect the Cimplicity driver to connect and poll the FieldServer for a range of valid addresses, and then proceed to poll for just the configured Points.
- From the start menu choose the "Point Control Panel", select edit and add the project to view.

NOTE: To log on the User name "ADMINISTRATOR" must be supplied.

- Use "Modbus ASCII Diagnostics" to connect to host and then read the register.

9.2.2 FieldServer with Intellution FIX as a Client

- Install Intellution FIX, choosing the MB1 Modbus ASCII I/O driver.
- Run from Start menu and choose “Intellution Fix”.
- Choose “System Configuration Utility”.
- Modify SCADA, add the MB1 Modbus ASCII I/O driver.
- Configure the Modbus ASCII Driver.
- Device is D11, select 5-digit address, add the FieldServer virtual Node ID to station address.
- Set up poll record.
- SAVE the configuration.
- Open “Startup”.
- Open “Mission Control” from the “Apps” menu and confirm Fix is polling.
- To display the data create a link in Fix draw, add link, data link.
- Give it a tagname, allow data entry, numeric entry and set enable option.
- If tag is not in database, select “Add”, choose “AR”. Then set output enable, device MB1, I/O address d11.
- Save the settings.
- Use “Quickview” from the “View” menu to confirm the reading of data without ??? appearing.
- Change the value and wait a few seconds to ensure the change occurred.

10 Reference

10.1 Data Types

If Node parameter Address_Type is set as ADU or PDU, then Data_Type must be specified as follows.

For Address_Type ADU:

Address range	Data_Type	Function Code (Write)	Function Code (Read)
1 - 65536	Coil	15	1
1 – 65536	Discrete_Input	n/a.	2
1 – 65536	Input_Register	n/a.	4
1 - 65536	Holding_Register	16	3

For Address_Type PDU:

Address range	Data_Type	Function Code (Write)	Function Code (Read)
0 - 65535	FC01	15	1
0 – 65535	FC02	n/a.	2
0 – 65535	FC04	n/a.	4
0 – 65535	FC03	16	3

For Address_Type Modicon_5digit:

When a Modbus address range is specified, a particular Data Type is implied. The defaults are shown below.

Address range	Data_Type	Function Code (Write)	Function Code (Read)
00001 - 09999	Coil	5,15	1
10001 - 19999	Discrete_Input	n/a.	2
30001 - 39999	Input_Register	n/a.	4
40001 - 49999	Holding_Register	6,16	3

10.2 Single Writes

If writing multiple registers, the write function will 16.

If writing multiple coils, the write function will 15.

If writing a single register, the write function will be 6 unless Write_FNC parameter is set to “Multiple”.

If writing a single coil, the write function will be 5 unless Write_FNC parameter is set to “Multi".