9210 & XPERT DATALOGGERS

MODBUS Manual

Part No. 8800-1147
Version 3.17
March 20, 2015

Sutron Corporation
22400 Davis Drive
Sterling, Virginia 20164
TEL: (703) 406-2800
FAX: (703) 406-2801
WEB: http://www.sutron.com/

Table of Contents

Introduction 4
Supported MODBUS Features 5
  Data Link Layer 5
  Application Layer 5
    Message Format 5
    Supported Function Codes (Slave) 6
    Message Handling 6
Installing modbus.sll 7
  Installation 7
Slave Configuration and Operation 8
  Communications Configuration (Slave Only) 8
    Serial Slave Configuration 8
    TCP Slave Configuration 9
  Identifying Registers, Coils, etc. (Slave only) 10
    Modbus Tag Block, “MBTag” 10
    Reserved Registers 14
  Sutron Function Code (Slave Only) 17
    Get File 17
    Send File 18
    Get Log 19
Master Configuration and Operation 21
  MBSensor 21
    MBSensor Values Configuration 22
    MBSensor Device Configuration 24
  MBOut 26
    Setting Up an MBOut Block 26

Table of Figures

Figure 1: The Modbus Control Panel Entry 8
Figure 2: Serial Slave Properties Dialog 8
Figure 3: TCP Slave Properties Dialog 10
Figure 4: Properties of the MBTag block 11
Figure 5: Typical Discrete Input Configuration 12
Figure 6: Typical Coil Configuration 13
Figure 7: Example Register Configuration 14
Figure 8: Properties of the Sensor Block 21
Figure 9: Configuring MBSensor "Values" 22
Figure 10: MBSensor Properties (Example) 23
Figure 11: Value #1 Properties (Example) 23
Figure 12: Value #2 Properties (Example) 24
Figure 13: MBSensor Port Properties 24
Figure 14: MBSensor Port Properties 25
Figure 15: Properties of the MBOut Block 27
Figure 16: "Value" properties for MBOut block 28

INTRODUCTION

Sutron's Xpert family of DCPs (both the 9210/B and the Xpert/2, hereafter referred to as the Xpert) have been designed to be easily expandable by adding additional software libraries, called Sutron Link Libraries (SLLs). One such library is modbus.sll, which adds the ability for the Xpert to communicate using the MODBUS protocol over serial communications links. This document is the user manual for modbus.sll. The following topics are discussed:

- How to install the library.
- What part of the MODBUS standard is supported by the library.
- How to configure the Xpert for MODBUS serial and/or TCP/IP communications.
- How data and operations inside the Xpert are identified as coils, discrete registers, and holding registers.
- How to configure the Xpert to read and write registers in MODBUS slave devices.

NOTE regarding TCP/IP communications:

The TCP/IP communication capabilities discussed in this manual apply only to Xpert2 and 9210B products. The original Xpert and 9210 (whose software versions stop at v2.x), do not have ethernet ports. Hence, they can only perform MODBUS serial communications.

SUPPORTED MODBUS FEATURES

This section discusses what portion of the MODBUS standard are supported by the modbus.sll library. To compare the features implemented by this library versus standard MODBUS, it may be helpful to refer to http://www.modbus.org for more information.

Data Link Layer

This library supports MODBUS messaging over asynchronous serial transmission, e.g., EIA/TIA-232 or EIA/TIA-485 in a master-slave arrangement, as well as over TCP/IP. Xpert can be either a MODBUS master or slave, depending on how the Xpert is configured. The Xpert can be both a master and slave on separate ports.

MODBUS RTU mode over serial uses 1 start bit, 8 data bits, 1 bit for parity completion, and 1 stop bit (2 stop bits, if no parity). MODBUS ASCII mode over serial uses 1 start bit, 7 data bits, 1 bit for parity completion, and 1 stop bit (2 stop bits, if no parity).

MODBUS communication over radios is supported through use of the hardware control lines. The user can specify delay and wake-up parameters to facilitate this mode of communication. RDI radios can be used for MODBUS messaging when this library is used in combination with RDI.sll, available separately.

MODBUS communication over modems is supported as long as the modem is manually configured prior to use. The modem must be configured for auto-answer, disabled hardware flow control, and DCE to DTE baud rate. Set the parity setting in the MODBUS configuration to “None”.

The following commands will configure the Sutron Voice Modem for MODBUS communication. Send these commands to the modem using a terminal program like HyperTerm set for 19200 baud. Then connect the modem to the Xpert.

Command	**Description**
AT&F	Reset modem to factory defaults.
ATS0=1	Set modem to auto-answer on 1 ring.
AT&K0	Disable DTE-DCE hardware flow control.
AT+IPR=19200	Set DCE to DTE baud rate to 19200.
AT&W	Write new settings to default configuration.

To verify the settings have been properly set, you can tytpe AT&V to view the settings for the active and stored profiles.

Application Layer

This section describes the application-layer standards that are supported by Xpert’s modbus.sll.

Message Format

The MODBUS serial communications standard specifies two different message formats, or modes, named RTU and ASCII.

The RTU message format represents data using a standard big-endian binary format.

The ASCII message format represents data using 2 ASCII characters to describe each binary byte in terms of its ASCII hexadecimal representation (e.g., the value 0xC9 would be communicated by sending the ASCII characters ‘C’ and ‘9’, as opposed to sending a byte having the value 0xC9, as would occur in RTU format).

The Xpert modbus.sll library supports RTU, ASCII, and TCP/IP message formats.

Supported Function Codes (Slave)

The following table identifies the functions that are supported by modbus.sll:

Code	Hex	Subcode	Hex
Read Coils	0x01		
Read Discrete Inputs	0x02		
Read Holding Registers	0x03		
Read Input Register	0x04		
Write Single Coil	0x05		
Write Single Register	0x06		
Diagnostic	0x08	Return Query Data	0x00
Diagnostic	0x08	Clear Counters	0x0A
Diagnostic	0x08	Return Bus Message Count	0x0B
Diagnostic	0x08	Return Bus Comm Error	0x0C
Diagnostic	0x08	Return Bus Exception Count	0x0D
Diagnostic	0x08	Return Slave Message Count	0x0E
Diagnostic	0x08	Return Slave Broadcast Count	0x0F
Diagnostic	0x08	Return Slave NAK Count	0x10
Diagnostic	0x08	ReturnSlave Busy Count	0x11
Diagnostic	0x08	Return Bus Char Overrun Count	0x12
Write Multiple Coils	0x0F		
Write Multiple Registers	0x10		
Sutron Function	0x41	Get File	“GF”
Sutron Function	0x41	Send File	“SF”
Sutron Function	0x41	Get Log	“GL”

Message Handling

Since MODBUS communications with Xpert read and write registers that are defined by the setup, the setup must be “locked” against change before these transactions can occur. This lock cannot occur when the setup is being changed by the user. Hence, when the user has the graphical setup page open (either remotely through Xterm, or locally at the station), MODBUS communications cannot complete successfully.

INSTALLING MODBUS.SLL

This section describes installation of the modbus.sll library.

Installation

To install the modbus.sll library, copy the file to the "\Flash Disk" subdirectory of your Xpert using Xterm. For more information on performing this file transfer, please refer to chapter 6 of the Xpert or 9210 user manual.

Once the library file has been transferred, reboot the Xpert. The library will load automatically after the Xpert reboots.

To uninstall the library, use Xterm to delete the file from the Flash Disk subdirectory. This can only be done when the Xpert application is not running (select "Exit App" from the Status tab).

In order for the modbus.sll library to load and operate correctly, the version of the modbus.sll file must be the same as the version of the application loaded into the Xpert. This is usually not a concern because the same versions of the sll and application are typically packaged together. Should the need arise to verify that the versions are the same, the version of the sll as it resides on the PC can be determined by looking at the file's properties (right-click on the file and select the "Version" tab). The version of the Xpert application is given by the application itself, at the top of the About dialog, which is accessed from the Status tab.

SLAVE CONFIGURATION AND OPERATION

This section describes how the Xpert is configured and operated as a MODBUS slave (i.e., responding to master software requests to read and write registers within the Xpert).

There are two steps to configure the Xpert for Slave operation:

1. Enable and configure a communications port for MODBUS.
2. Identify Registers, Coils, etc., using MBTag blocks.

Communications Configuration (Slave Only)

MODBUS slaves are created and configured using the "Modbus Slaves" entry on the Setup tab. To create a new MODBUS slave, select the entry, and press the "New" button. Another dialog will prompt the type of slave (serial or TCP), and then show the configuration dialog. Alternatively, to configure an existing slave, highlight it and press the "Edit" button.

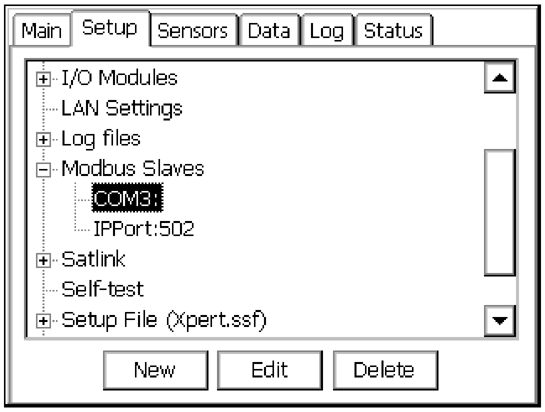
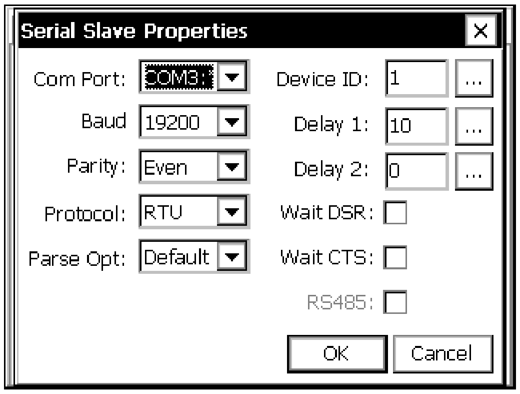


Figure 1: The Modbus Control Panel Entry

Serial Slave Configuration

The dialog shown below is used to configure serial slaves.



Com Port: COM3:
Baud 19200
Parity: Even
Protocol: RTU
Parse Opt: Default
Device ID: 1
Delay 1: 10
Delay 2: 0
Wait DSR: ☐
Wait CTS: ☐
RS485: ☐

Figure 2: Serial Slave Properties Dialog

The modbus serial communications properties set in this dialog are defined as follows:

Com Port	This identifies the comm. port to configure for MODBUS communications. The possible selections are COM2 through COM9. The default is COM3.
DeviceID	This identifies the address of the Xpert on a MODBUS serial bus. The Xpert will process all messages it receives that contain this address in the device id field of the message (as well as "broadcast" messages, i.e., device id 0). This device id applies only to this particular modbus slave, not to the Xpert as a whole, allowing different slaves to have different device ids.
Baud Rate	This identifies the baud rate to use for MODBUS communications over the selected comm port.
Protocol	This identifies the message format and framing protocol to be used during serial communications. The possible selections are either RTU or ASCII. The default is RTU.
Parity	This identifies the parity setting of the comm port. The possible selections are even, odd, and none. Even is the default parity.
Delay 1	This identifies the number of milliseconds to wait after asserting RTS before starting data transmission. The default is 10ms.
Delay 2	This identifies the number of milliseconds to wait after data transmission is complete before de-asserting RTS. The default is 0ms.
Wait DSR	Specifies whether the DSR (data-set-ready) signal is monitored for output flow control. If this box is checked and the DSR input control line is low, output is suspended until DSR is high again.
Wait CTS	Specifies whether the CTS (clear-to-send) signal is monitored for output flow control. If this box is checked and the CTS input control line is low, output is suspended until CTS is high again.
Parse Opt.	Use this to select the modbus parser. Set this to "Default" unless you have installed custom modbus software that provides other parser types.

Note: Be sure to select a comm port that IS NOT IN USE by the Coms Manager. To do this, select the "Coms" entry of the control panel tree on the Setup tab, press "Edit", and verify that the mode of the port you selected for MODBUS is set to "None".

TCP Slave Configuration

The dialog shown below is used to configure TCP slaves.

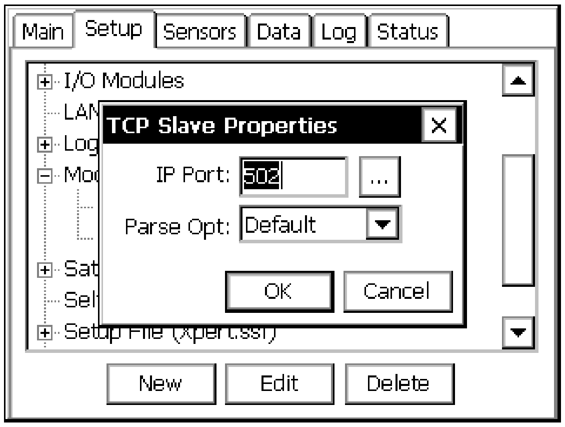


Figure 3: TCP Slave Properties Dialog

The TCP Slave Properties set in this dialog are defined as follows:

IP Port	This identifies the IP port to use when listening for messages from MODBUS masters. The default is 502 (which is the standard IP port allocated to MODBUS; use caution when changing this to another value).
Parse Opt.	Use this to select the modbus parser. Set this to "Default" unless you have installed custom modbus software that provides other parser types.

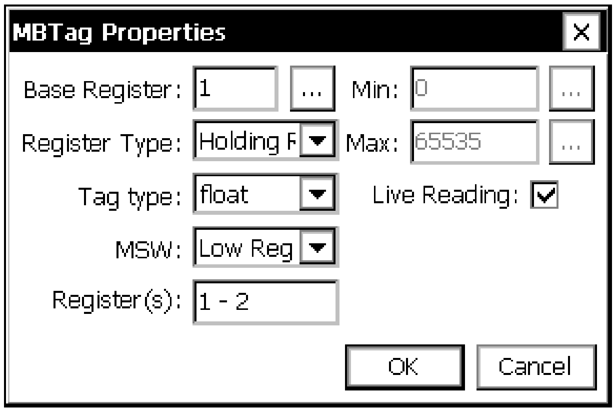
Identifying Registers, Coils, etc. (Slave only)

This section describes how the Xpert's data and operations have been mapped into the MODBUS concepts of coils, discrete inputs, input registers, and holding registers, when the Xpert operates as a MODBUS slave.

Commands from the master device to read and/or write coils and registers inside the Xpert either refer to data points in the graphical setup defined by a modbus tag block, or to predefined state and operational control variables. Each of these mappings is discussed in more detail in the following sections.

Modbus Tag Block, "MBTag"

The modbus tag block, or "MBTag", is the second visual component added to the Xpert by the modbus.sll library, and is used to map MODBUS coils, discretes, and registers to data points within the graphical setup. Each block has a number of properties that determine how the block responds to data read and write requests. The properties dialog for the MBTag block is shown below:


MBTag Properties

Base Register:	1	Min:	0
Register Type:	Holding R	Max:	65535
Tag type:	float	Live Reading:	✓
MSW:	Low Reg		
Register(s):	1 - 2		

Figure 4: Properties of the MBTag block

The properties of the MBTag block are defined as follows:

Base Register	Identifies the register number of the data contained in the block. When multiple registers are needed to represent the data (as in the case of floats and ints), then this number is the base, or lowest, number of the register pair. Note that the Xpert starts numbering registers at 1, not 0. Some MODBUS masters start numbering at 0. In that case, configure the master to retrieve the register number one less than you assign in the Xpert.
Register Type	Identifies the type of register this block represents. The possible selections are: Discrete Input, Coil, Input Register, and Holding Register. The Coil type is typically selected when the input block (i.e., the block connected to the left of the MBTag block in the graphical setup) is a BinOut. The Discrete Input is typically selected when the input block is a BinIn. The Holding Register type is typically selected when the input block is a Const. Either the Input Register type or Holding Register type is typically selected for all other scenarios.
Tag Type	Identifies the data type of the tag. This field is enabled only when the Register Type is Input or Holding Register. The possible selections are: short, ushort, int, uint, and float. The short and ushort types are single 16-bit registers. The int, uint, and float types use two 16-bit registers. For example, if the Tag Type is float and the Base Register valaue is set to 1, then both registers 1 and 2 are used represent the value of the tag.
MSW	For multi-register tag types (int, uint, and float), determines which register is used to contain the most significant word (MSW) of the tag's value. When set to "Low Reg", the lower register of the register pair is used to contain the MSW. For example, given a Base Register of 1, a Register Type of Input Register, a Tag Type of float, and a MSW setting of Low Reg, then a master request for register 1 returns the MSW of the tag's value in IEEE 32-bit float format. A

	request for register 2 in this scenario returns the LSW of the tag's float value. This item is disabled when the tag uses only one register (coil, discrete, ushort, etc.).
Register(s)	Displays the register numbers used to represent the tag. This field is read-only.
Min	For input and holding registers of tag type short or ushort, Min and Max are used to scale the block's data into the integral range 0 – 65535 (the range of the data type used to communicate register data back to the Modbus master device). Typically, you enter into Min the lowest value you expect this block to contain.
Max	For input and holding registers of tag type short or ushort, Min and Max are used to scale the block's data into the integral range 0 – 65535 (the range of the data type used to communicate register data back to the Modbus master device). Typically, you enter into Max the highest value you expect this block to contain.
Live Reading	When this checkbox is checked, a "live reading" is performed when a request to read or write the data is received. This causes the inputs and outputs to the MBTag block to perform their normal processing (just as would be the case in a scheduled execution due to a Measure block, for example).

Discrete Inputs

A Discrete Input is a 1-bit data point that may only be read. The analogous object in the Xpert graphical setup is the BinIn block. Therefore, use an MBTag block with its type set to "Discrete Input" to identify a BinIn block as a discrete input. An example configuration appears in the figure below.

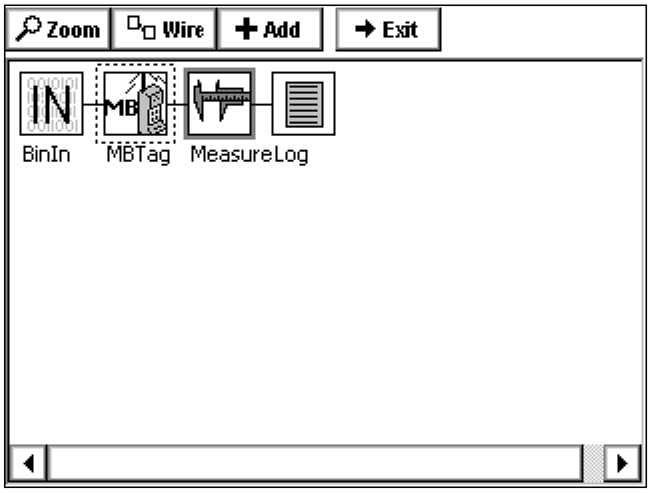


Figure 5: Typical Discrete Input Configuration

Reading the discrete input results in a non-zero value (typically 1) if the input is hi (+5V), or 0 if the output is low (0V).

LiveReading is typically not checked for discrete inputs, unless you desire that a read of the discrete also cause the chain of blocks containing the MBTag block be executed.

In the example configuration, a Measure block exists to poll and log the value of the discrete. This is not required for a MODBUS query of the discrete to succeed. Note that a live reading of the block in this configuration will sample the input before returning the value, but will not cause the value to be logged (Measure blocks do not allow readings to be “pushed” through them).

Discrete input readings are not affected by the values contained in Min and Max.

Coils

A Coil is a discrete data point that may be both read and written. The analogous object in the Xpert graphical setup is the BinOut block. Therefore, use an MBTag block with its type set to “Coil” to identify a BinOut block as a coil. An example configuration appears in the figure below.

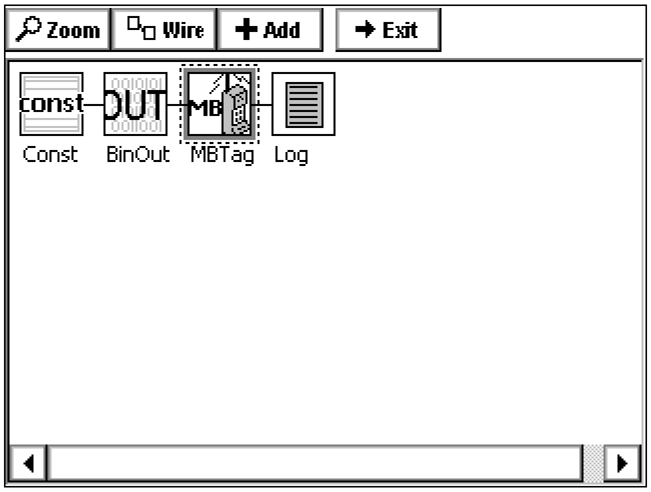


Figure 6: Typical Coil Configuration

The Const block in the example configuration is necessary since the BinOut block is an output block and, therefore, cannot exist without an input. The Const block ends-up taking-on whatever value is written to the coil.

Reading the coil results in a non-zero value (typically 1) if the output is ON (0V), or 0 if the output is OFF (+5V). When writing to the coil, write a 1 to turn the output ON or a 0 to turn the output OFF.

LiveReading must be checked in order to set the output immediately. Otherwise, the output will not change until the next execution of the BinOut block, which normally occurs due to some other in-line active block. In the example configuration, LiveReading is checked (as evidenced by the dark shading around the block). Since there is no other active block in this configuration (e.g., a Measure block), the output changes only when a write occurs.

Since LiveReading applies to both reading and writing, the chain of blocks containing the MBTag block are executed even when reading the coil while the LiveReading setting is checked. This typically doesn’t change the value of the coil, but can result in unexepected side-effects, depending on what other blocks may be connected. For example, a read of the coil in the example configuration has the unexpected side-effect of logging the current value.

Coil readings and writes are not affected by the values contained in Min and Max.

Registers

An Input Register is a read-only 16-bit data point while a Holding Register is a 16-bit data point that can read or written. Any data point in the Xpert graphical setup can be treated as one of these registers. An example configuration appears in the figure below.

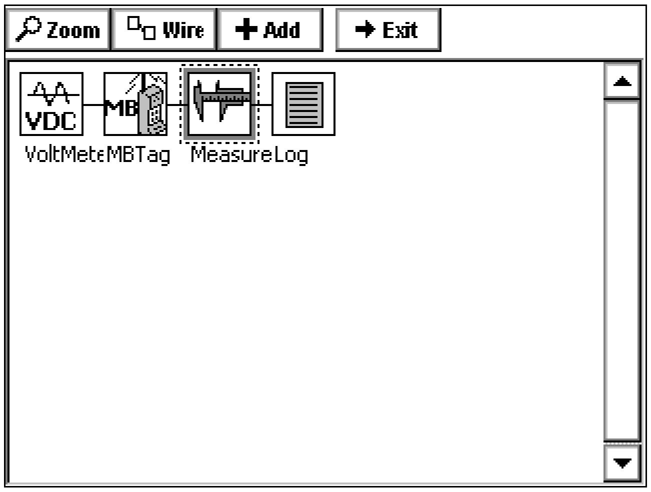


Figure 7: Example Register Configuration

When the Tag Type is set to short or ushort, reading the register results in a value between 0 and 65535. The Min and Max properties of the MBTag block are used to scale the value into this range so that values in other ranges may be represented. For example, if the expected value of the output of the VoltMeter block is –50.0 and +50.0, then enter –50.0 as Min and +50.0 as Max. A VoltMeter output of –50 will read as 0, an output of 0 will read as 32768, and an output of +50.0 will read as 65535. The converse is true when writing to a holding register, i.e., writing a value of 65535 to the register results in the block taking on the value of +50.0, and so on.

The LiveReading property should be checked if you desire that a read (or write, in the case of a holding register) also cause the chain of blocks containing the MBTag block to be executed (for example, make a log entry, if a log block is attached with no Measure in between).

In the example configuration, a Measure block exists to poll and log the value of the register. This is not required for a MODBUS read or write to succeed.

Reserved Registers

Holding registers in the range 1001 –1099 (4:01001 to 4:01099) and 2001 – 2011 (4:02001 to 4:02011) have been reserved for the purpose of setting and getting Xpert state data, controlling the operation of the Xpert, and for retrieving log data.

The following table identifies the actions that occur when reserved registers 1001 – 1099 are read or written.

Reg #	Register Description	Read	Write
4:01001	Hour of current time.	Returns hour in range 0 – 23.	Sets hour. Use range 0 – 23.
4:01002	Minute of current time.	Returns minute in range 0 – 59.	Sets minute. Use range 0 – 59.
4:01003	Second of current time.	Returns second in range 0 – 59.	Sets second. Use range 0 – 59.
4:01004	Year of current data.	Returns year.	Sets year.
4:01005	Month of current date.	Returns month in range 1 – 12.	Sets month. Use range 1 – 12.
4:01006	Day of current month.	Returns day in range 1 – 31.	Sets day. Use range 1 – 31.
4:01007	Recording status.	Returns 1 to indicate recording is on, 0 to indicate recording is off.	Write 1 to turn recording on, write 0 to turn recording off.
4:01008	Reset unit.	Returns 0.	Write 1 to reset the unit.
4:01009	Transmission protocol.	Returns 0 to indicate RTU protocol. Returns 1 to indicate ASCII protocol.	Write 0 to set protocol to RTU. Write 1 to set protocol to ASCII. The response to this message will be in the original protocol.

Reading Log Data

The second set of reserved registers (2001-2011) can be used to read entries from the default log (SSP.LOG). These registers also allow the master equipment to notify the Xpert which log entries have been successfully read (i.e., to *acknowledge* logged data that has been read). This allows the master equipment to implement a read and acknowledge scheme that ensures the ability to read all logged data, even when the Xpert or master station reset unexpectedly, with minimal redundant reporting. State data is maintained across power cycles, so acknowledged data is never reported again.

The following table identifies the registers used to read and acknowledge logged data.

Reg #	Contents	Description	R/W
4:02001	Index	Index of oldest unacknowledged log entry. Starts at 0. For example, "0" represents the oldest unacknowledged log entry, "1" represents the next oldest, and so on..	R/W
4:02002	Hour	Log entry time stamp	R
4:02003	Minute		

4:02004	Second		
4:02005	Year		
4:02006	Month		
4:02007	Day		
4:02008	Entry ID	This register contains the logged entry label converted to an unsigned 16-bit integer. Hence, when labeling logged data, use ASCII numerals (“1”, “2”, etc.).	R
4:02009	Log Entry Value	Value of log entry in 32-bit floating point format.	R
4:02010			
4:02011	Quality	Quality of data in the log. 0 = GOOD, 1 = BAD, 2 = UNDEFINED.	R

To read an entry from the log, read registers 2001 – 2011 in a single multi-register read (note: reads of individual registers will fail). The Index register is automatically incremented after a read, so that the master station can tell how many data points it has read since its last acknowledgement.

The master station can write to the Index register as desired, in order to request previously reported, but as yet unacknowledged log data. For example, if the master station resets and notes on the next read of logged data that the index is above 0, the master can write 0 to the Index register in order to pick-up with the data point the master last acknowledged.

The master station acknowledges data points by writing the value of the data to acknowledge to registers 2002 – 2008. The data from this write is not stored in the registers; it is simply used to identify the data point(s) being acknowledged.

When the master station attempts to read beyond the end of the log, registers 2002 – 2011 contain all zeroes, and register 2001 contains the total number of unacknowledged log entries.

For example, assume there are two unacknowledged events…

- Write 0 to register 2001
- Read 2001/2011. Register 2001 contains 0, so the oldest unacknowledged event is returned. The Xpert increments register 2001 to 1 after transmission.
- Read 2001/2011. Register 2001 contains 1, so the next oldest unacknowledged event is returned. The Xpert increments register 2001 to 2 after transmission.
- Read 2001/2011. Register 2001 contains 2, but there is no corresponding unacknowledged event. Registers 2002 through 2011 contain 0. The Xpert does not increment register 2001. Another attempt to read will get identical information unless data has been logged subsequently.

Here are some important notes regarding this method of reading the log:

- Works only with ssp.log.

- Log lables (entered in log block in graphical setup) must be unique to each sensor and must be numerals less than 65536.
- EzSetup must not be used to log data when using dthis method to retrieve data from the log.
- When reading a log entry, registers 2001 – 2011 must be read as part of a single multi-register read. Reading anything less in this range will result in an exception response.
- The master station must acknowledge data read before the data from the last acknowledge is overwritten due to log wrap, as well as before the Index counter exceeds 65535.
- The master station's response timeout may increase and need to be adjusted when the Xpert's log is large and full.

Sutron Function Code (Slave Only)

This section describes the use of the Sutron function code to get and send files and logs over MODBUS.

The data portion of a packet carrying the Sutron function code (function code 65, or 0x41) contains a subcode and associated parameters to define the particular function to perform. The subcode and its parameters are expressed using ASCII characters. The functions that each sub-code performs are defined to be stateless, and the burden of the transport is on the master station.

The following text defines the subcode functions, parameters, and responses.

Get File

The Get File subcode is used to retrieve a file from Xpert. The format of the command is as follows:

```
GF,filename,startpos,numbytes
```

The file is assumed to be in \Flash Disk unless a full path is given. There can be no commas in the path/file name.

File start positions (startpos) are 0-based.

The numbytes value refers to the number of data bytes of the requested file to include in the response, not the number of bytes to store in the return packet. Since the master station or transport medium may be limited in the number of bytes that can be handled in a single packet, the numbytes value should be sized small enough to leave room for header and CRC information, as well as the translation to ASCII if that is the selected protocol (the ASCII protocol uses two bytes to represent every data byte).

If numbytes is *, the entire file will be returned.

The format of the reply is as follows:

```
GFR,status,totalbytes,start,numbytes,data
```

The value of status can be any of the following values:

Value	Description
0	Ok.

1	File not found.
2	Get beyond file end.
7	Command format error

Totalbytes tells the master station how long the file is. The master station should issue repeated GetFile commands until it has received totalbytes amount of data. Retries are the responsibility of the master station. If totalbytes changes in the reply, the file has changed and the get should be restarted. If a reply is missing, the master station must request the data again.

The numbytes value in the GFR response does not include the comma preceding the data, and refers to the number of data bytes from the file that are being returned, not the number of packet bytes used to store the response (which would be twice the number of data bytes when ASCII protocol is selected).

Example:

```
command:    GF,Myname.ssf,0,230
reply:      GFR,0,34210,0,230,data
command:    GF,Myname.ssf,230,230
reply:      GFR,0,34210,230,230,data
…
command:    GF,Myname.ssf,34040,170
reply:      GFR,0,34040,170,data
```

GetFile can be used for actual files (such as the setup file) or for virtual files. The following virtual files can be sent:

1. Current data. This data is requested by requesting the file "curdata.txt". The Xpert software collects the last value of all coms tags in the system and sends this data as a text file with the following format: Date,Time,Label,Value,Units,Quality. Each line is terminated by a CR/LF. Date and Time are expressed as in “01/31/2004” and “20:47:52”, respectively. Quality is a single character defining the quality of the data where “G” = Good, “B” = Bad, and “U” = Unknown.
2. Status information. This data is requested by requesting the file "status.txt". The Xpert software sends the contents of the status screen at the time of the request.
3. Directory contents. The contents of a directory can be retrieved by requesting the directory by name. The contents of the directory are sent as a text file formatted as follows: Name,Size,Date. Each line is terminated by a CR/LF.
4. Sensor Data. This data is requested by requesting the file sensors.txt. The data in the file is the same as that displayed on the view sensors page. Tab characters are used to denote column breaks. Each line is terminated by a CR/LF.

Send File

The Send File subcode is used to send a file to Xpert. The format of the command is as follows:

```
SF,filename,bytepos,numbytes,data
```

The file is assumed to be destined for \Flash Disk unless a full path is given. There can be no commas in the path/file name.

File positions (bytepos) are 0-based.

The numbytes value refers to the number of data bytes of the file being sent, not the number of bytes in the data portion of the packet (which uses two bytes for every single data byte when ASCII protocol is selected).

The format of the reply is as follows:

```
SFR,status,bytepos,numbytes
```

The value of status can be any of the following values:

Value	Description
0	Ok.
3	Write failed.
4	Gap at end of file.
5	Failed to open for write (file may be in use).
7	Command format error

Example:

```
command:    SF,Myname.ssf,0,230,data
reply:      SFR,0,0,230
command:    SF,Myname.ssf,230,230,data
reply:      SFR,0,230,230
…
command:    SF,Myname.ssf,34200,9,data
reply:      SFR,0,34200,9
```

Again, the master station is responsible to monitor each reply to make sure that the proper number of bytes have been transferred and that there were no errors of any kind in the process. If errors are detected, the transfer must be restarted.

Get Log

The Get Log subcode is used to retrieve log data from the Xpert. The format of the command is as follows:

```
GL,logfilename,datetime,recordID,numbytes
```

The log file is assumed to be in \Flash Disk unless a full path is given. There can be no commas in the path/file name.

The datetime value must be in the following format: MM/DD/YYYY HH:MM:SS.

The numbytes value refers to the number of data bytes from the log entry to include in the response, not the number of bytes to store in the return packet. Since the master station or transport medium may be limited in the number of bytes that can be handled in a single packet, the numbytes value should be sized small enough to allow for header and CRC information, as well as translation to ASCII if that is the selected protocol (the ASCII protocol uses two bytes to represent every data byte).

If numbytes is *, all log records found will be returned.

Regardless of the requested numbytes, only complete log records are returned.

The format of the reply is as follows:

```
GLR,status,recordID,numbytes,data[numbytes,data]
```

The value of status can be any of the following values:

Value	Description
0	Ok.
1	File not found.
6	Record not found.
7	Command format error

The datetime value in the response message is the datetime of the returned record and, therefore, may be different from the datetime in the GetLog command statement.

The data to the end of the file can be read by leaving datetime at the desired starting point and incrementing recordID until the status indicates record not found. The [numbytes,data] represents an additional record of data if there is room in the message.

The times in the Xpert log are not necessarily chronological. There can be 5 records all with the time 12:10:00 followed by a record with time 12:10:10, and then some more at 12:10:00. Since datetime will always point to the first record with the requested datetime, the subsequent records can be retrieved without changing datetime to insure all the data is retrieved.

Example:

```
command:    GL,SSP.log,10/07/2003 15:50:00,0,80
reply:      GLR,0,0,38,10/07/2003,15:51:00,VBAT,13.16,Volts,G
            37,10/07/2003,15:51:10,A,10.89,5.2,-25.4
command:    GL,SSP.log,10/07/2003 15:50:00,2,80
reply:      GLR,0,2,37,10/07/2003,15:54:00,C,10.89,5.2,-25.4,0
```

Wrapped for illustration purposes only.

The GLR response will contain as many log records as can fit into the response.

The numbytes value in the GLR response does not include the comma preceding the data, and refers to the number of data bytes from the log that are being returned, not the number of packet bytes used to store the response (which would be twice the data bytes when ASCII protocol is selected).

The Xpert has two types of log records. The first type contains date, time, sensor, value, units, and quality, and comes from a standard log block. The other type contains date, time, id, value1, …, and valuen, and comes from EZSETUP sensors with a logid specified in the measurement properties.

MASTER CONFIGURATION AND OPERATION

This section describes how the Xpert is configured and operated as a MODBUS master (i.e., initiating requests to read and write registers in another device, e.g., a sensor).

The Xpert uses the MBSensor and MBOut blocks to operate as a MODBUS master. There is no need to configure a comm or tcp port in either the Modbus or Coms control panel entries on the Setup tab for master operation. In fact, doing so will interfere with the configuration done in the properties of each of the blocks.

There is no problem configuring an Xpert to operate as both a master and a slave at the same time, as long as each function uses a separate port. For example, if a slave port is defined on COM2, then any MBSensor and/or MBOut blocks must use a port other than COM2.

When MBSensor and MBOut blocks are configured to use the same com port, then each must be scheduled so that they don't attempt to use the port at the same time.

MBSensor

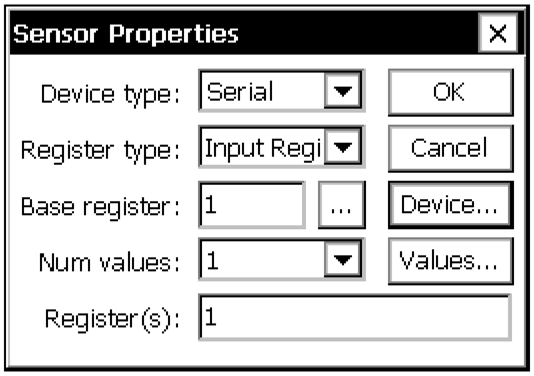
The MBSensor block is used to read registers from a MODBUS slave device and translate those registers into output values (up to 20).

The MBSensor block is not scheduled. This means that a Measure or other kind of "active" bock must be connected to the MBSensor block in order for the block to execute.

Each MBSensor block is assigned either a tcp or serial comm port in the block's properties dialog (see below). The same port can be used as other MBSensor blocks, as long as the blocks are scheduled to execute at different times.

Note: make sure any serial comm port assigned to the MBSensor block is not being used by the Coms Manager. To do this, go to the Coms control panel entry on the Setup tab, press Edit, and make sure the mode for the port is set to "None".

The properties dialog for the MBSensor block appears below.


Sensor Properties

Device type: Serial

Register type: Input Regi

Base register: 1

Num values: 1

Register(s): 1

Figure 8: Properties of the Sensor Block

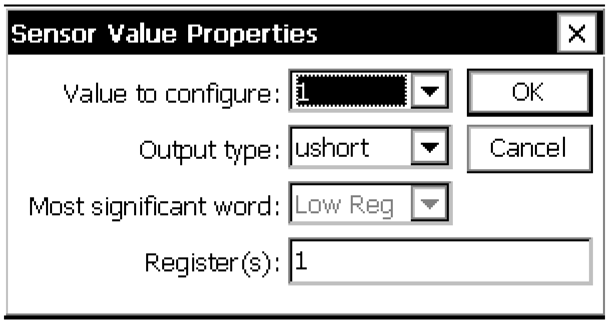
The properties of the MBSensor block are defined as follows:

Device type Identifies the type of MODBUS device to target, either Serial (for serial

	communications, or Tcp for ethernet communications).
Register Type	Identifies the type of register this block will read. The possible selections are: Discrete Input, Coil, Input Register, and Holding Register.
Base Register	Identifies the register to read. When multiple registers will be read in order to produce the block's outputs, then this number is the base, or lowest, number of the register set. Note that the Xpert starts numbering registers at 1, not 0. Some MODBUS slaves start numbering at 0. In that case, configure the Xpert to read the register number that is one higher than the slave's register number.
Num Values	Identifies the number of outputs from this block. An MBSensor block can define up to 20 output values. The "Values…" button opens the dialog used to configure how the registers read are used to produce the output values.
Register(s)	Displays the register numbers that will be read to produce the block's output values. This field is read-only.

MBSensor Values Configuration

Pressing the "Values…" button brings up the following dialog:


Sensor Value Properties

Value to configure: 1

Output type: ushort

Most significant word: Low Reg

Register(s): 1

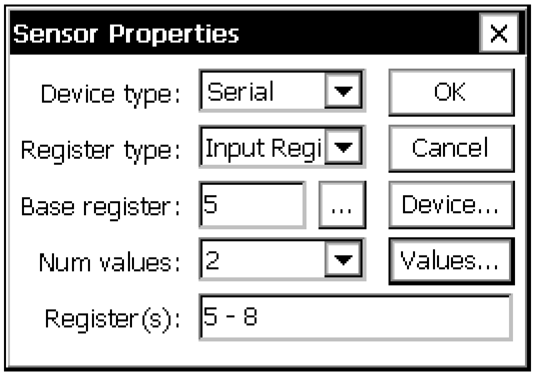
Figure 9: Configuring MBSensor "Values"

The Value Properties dialog is used to configure how registers read from the sensor are translated into outputs. The properties are defined as follows:

Value to configure	Identifies the value number to configure. This ranges from 1 to the total number of values entered on the MBSensor properties page. Select each value in turn and set the output type and most significant word, if applicable, for each.
Output type	Identifies the type of output (short, ushort, int, uint, or float).
Most significant word	For multi-register output types (int, uint, and float), determines which register is used to contain the most significant word (MSW) of the output value. When set to "Low Reg", the lower register of the register pair is used to contain the MSW. This item is disabled when the output uses only one register (coil,

	discrete, ushort, or short.).
Register(s)	Identifies the registers that are used to produce the selected output. This field is read-only.

Here's an example of how the properties would be configured to read a sensor having an integer at input registers 5 and 6, and a float at input registers 7 and 8. The MBSensor Properties dialog would have Register Type set to "Input Register", Base Register set to 5, and Num Values set to 2, as in the following figure:


Sensor Properties

Device type: Serial

Register type: Input Regi

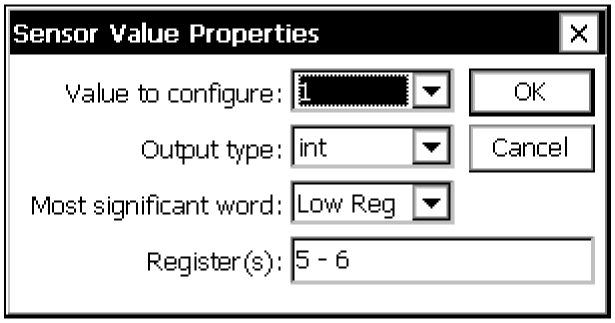
Base register: 5

Num values: 2

Register(s): 5 - 8

Figure 10: MBSensor Properties (Example)

Clicking the "Values…" button brings up the Sensor Value Properties dialog:


Sensor Value Properties

Value to configure: 1

Output type: int

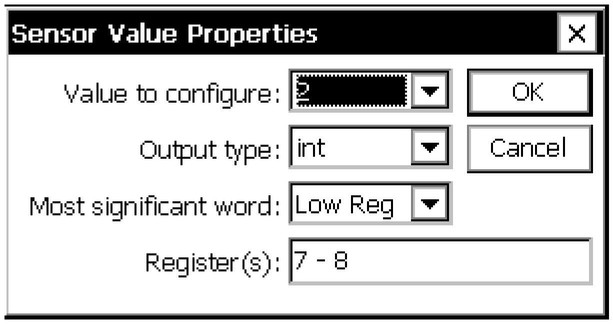
Most significant word: Low Reg

Register(s): 5 - 6

Figure 11: Value #1 Properties (Example)

From this dialog we see that output value 1 has type "int", and that the most significant word of the int will come from the lowest register of the register pair. The Register(s) field explains that the 2 registers used to produce the output are 5 and 6.

The figure below shows what the dialog would look like after changing the "Value to configure" selection to 2:


Sensor Value Properties

Value to configure: 2

Output type: int

Most significant word: Low Reg

Register(s): 7 - 8

Figure 12: Value #2 Properties (Example)

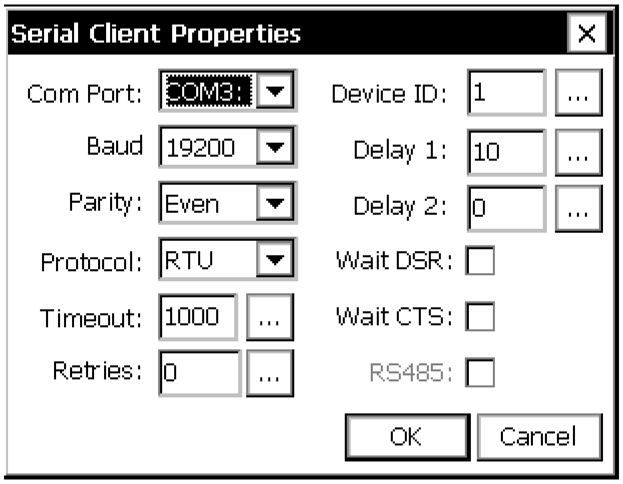
The dialog now shows us that output value 2 has type "float", and that the most significant word of the float will come from the lowest register of the register pair (i.e., register 7). The Register(s) field explains that the 2 registers used to produce the output are 7 and 8.

MBSensor Device Configuration

Pressing the "Device…" button on the MBSensor Properties dialog brings up a dialog used to configure the selected device (either Serial or Tcp):

Serial Master ("Client") Device Configuration

The following dialog is used to configure the serial communication properties for a MODBUS master over a serial comm port:


Serial Client Properties

Com Port: COM3:

Baud 19200

Parity: Even

Protocol: RTU

Timeout: 1000

Retries: 0

Device ID: 1

Delay 1: 10

Delay 2: 0

Wait DSR: ☐

Wait CTS: ☐

RS485: ☐

Figure 13: MBSensor Port Properties

The properties set in this dialog are defined as follows:

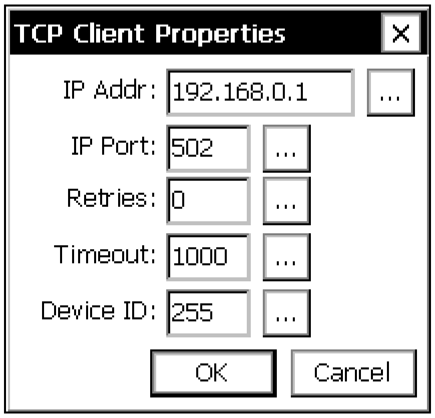
Com Port	Identifies the comm port to use for this MBSensor's MODBUS communications. The possible selections are COM2 through COM9.
Baud Rate	Identifies the baud rate to use on the selected comm port.
Delay 1	Identifies the number of milliseconds to wait after asserting RTS before

	starting data transmission. The default is 10ms.
Delay 2	Identifies the number of milliseconds to wait after data transmission is complete before de-asserting RTS. The default is 10ms.
Retries	Identifies the number of times to retry a communication if the first one fails.
Protocol	Identifies the message format and framing protocol to be used during serial communications. The possible selections are RTU and ASCII. The default is RTU.
Parity	Identifies the parity setting of the comm port. The possible selections are even, odd, and none. Even is the default parity.
Wait DSR	Specifies whether the DSR (data-set-ready) signal is monitored for output flow control. If this box is checked and the DSR input control line is low, output is suspended until DSR is high again.
Wait CTS	Specifies whether the CTS (clear-to-send) signal is monitored for output flow control. If this box is checked and the CTS input control line is low, output is suspended until CTS is high again.
Timeout	Identifies the number of milliseconds the Xpert will wait for a response from the device.

Note: Be sure to select a comm port that IS NOT IN USE by the Coms Manager. To do this, select the "Coms" entry of the control panel tree on the Setup tab, press "Edit", and verify that the mode of the port you selected for MODBUS is set to "None".

Tcp Master ("Client") Device Configuration

The following dialog is used to configure the TCP/IP communication properties for a MODBUS master over a Tcp port:


TCP Client Properties

IP Addr: 192.168.0.1

IP Port: 502

Retries: 0

Timeout: 1000

Device ID: 255

Figure 14: MBSensor Port Properties

The properties set in this dialog are defined as follows:

IP Addr	Identifies the IP address of the MODBUS slave.
IP Port	Identifies the IP port of the MODBUS slave.
Retries	Identifies the number of times to retry a communication if the first one fails.
Timeout	Identifies the number of milliseconds to give the slave device to respond before declaring the slave did not respond.
Device ID	The device ID (or "unit" ID) of the target device. This is typically not used in Modbus TCP, but is made available as some Modbus slaves require it to be set to a specific value (typically either 0 or 255). Setting this field is also required when communicating with a serial bridge, where the device ID is used once the message is transferred to the serial bus.

MBOut

The MBOut block is used to write registers in a MODBUS slave device. The block allows you to define local modbus registers that you want to write into the slave device's registers.

The MBOut block is not scheduled. This means that a Measure or other kind of "active" bock must be connected to the MBSensor block in order for the block to execute.

Each MBOut block is assigned a comm port in the block's properties dialog (see below). The same port can be used as other MBOut blocks, as long as the blocks are scheduled to execute at different times.

Note: make sure any serial comm port assigned to the MBOut block is not being used by the Coms Manager. To do this, go to the Coms control panel entry on the Setup tab, press Edit, and make sure the mode for the port is set to "None".

Setting Up an MBOut Block

The inputs and outputs of the MBOut block *do not* contain the data to be sent to the modbus slave device. The block's input is used to enable the output. As long as the input is non-zero, the block will continue to function. The block's output contains a modbus communication result code. When this value is 0, the communication succeeded without error. Otherwise, the code is the exception response code returned from the modbus slave device.

A combination of MBOut block properties and MBTag blocks are used to identify the registers and data to be written to the MODBUS slave device. MBTag blocks are used to define the data points that serve as registers in the Xpert/9210. These registers are referred to as "local" registers. The properties allow you to define which local registers should be written to which registers in the slave device.

MBOut Properties

The properties dialog for the MBOut block appears below.

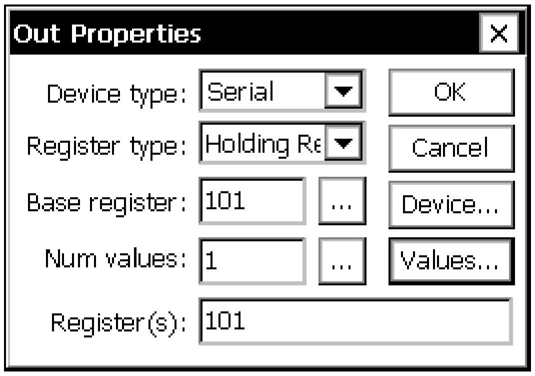


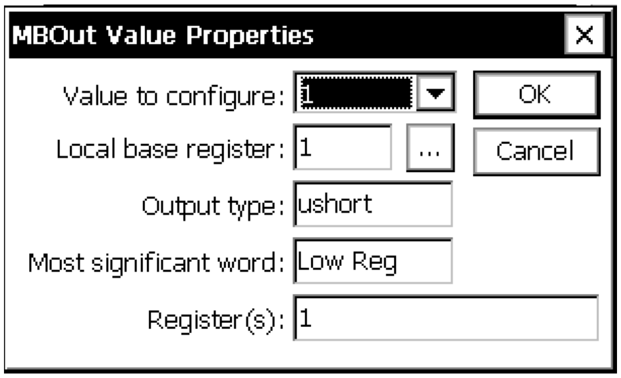
Figure 15: Properties of the MBOut Block

The properties of the MBOut block are defined as follows:

Device Type	Identifies the type of MODBUS device to target, either Serial (for serial communications, or Tcp for ethernet communications).
Register Type	Identifies the type of register this block will write. The possible selections are: Coil and Holding Register.
Base Register	Identifies the register to write. When the output type requires that multiple registers are written, then this number is the base, or lowest, number of the register set. Note that the Xpert starts numbering registers at 1, not 0. Some MODBUS slaves start numbering at 0. In that case, configure the Xpert to write the register number that is one higher than the slave's register number.
Num Values	Identifies the number of values to write to the slave device. Each value corresponds to a value in the current setup marked by an MBTag that you want to write into the target device. The "Values…" button opens the dialog used to configure which local registers should be written into which target registers. Note that Num Values is NOT necessarily the same as the number of registers that will be written, since some value types (e.g. ints and floats) require more than one register be written.
Register(s)	Displays the register numbers that will be written in the target device (slave). This field is read-only and is set based on the current configuration. Use it to verify you have configured the setup correctly to write the intended slave registers.

MBOut Values Configuration

Pressing the "Values…" button on the MBOut properties dialog brings up the MBOut Values dialog (see below). This dialog is used to select local registers to be written into the target slave registers. The dialog in the figure shows the first value will come from local base register 1, which has output type "ushort" (making "Most significant word" irrelevant). If you enter a number for local register that does not yet have a corresponding MBTag, an error will appear in the "Register(s)" field saying so.



Value to configure: 1
Local base register: 1
Output type: ushort
Most significant word: Low Reg
Register(s): 1

Figure 16: "Value" properties for MBOut block

The “Value” properties of the MBOut block are defined as follows:

Value to configure	Identifies the value number to configure. This ranges from 1 to the total number of values entered on the MBOut properties page. Select each value in turn and set the local base register for each. Note this requres that you have previously inserted MBTag blocks in the setup to define “local” registers.
Local base register	Identifies the local register from which to obtain the data value to write into the slave. When more than one register is required for the value (e.g., when the value is of int, unit, or float type), then this number is the base, or lowest, register of the pair. Regardless, this is the number you will have entered into register number field on the MBTag block. Note: the type of the local register must match the register type set in the MBOut Properties.
Output type	Identifies the type of data to write (short, ushort, int, uint, or float). This field is read-only and is populated with the information obtained from the MBTag block.
MSW	For multi-register output types (int, uint, and float), determines which register is used to contain the most significant word (MSW) of the output value. When set to “Low Reg”, the lower register of the register pair is used to contain the MSW. This field is read-only and is populated with the information obtained from the MBTag block.
Register(s)	Displays the register numbers that will be written in the slave device. This field is read-only. Use it to verify you have configured the setup correctly to write the intended slave registers.

MBOut Device Configuration

Pressing the “Device…” button on the MBOut Properties dialog brings up a dialog used to assign and configure a communications port (serial or Tcp) used for this block’s MODBUS communication. The settings are the same as for the MBSensor block. See MBSensor Port Configuration.