



Dynamic Rating, Monitoring, Control and Communications

Modbus User Guide

Revision History

6-Mar-01:	M. Witte:	Renamed and totally revised
24-Nov-00:	P. Stewart:	Totally revised
17-Mar-00:	P. Stewart:	Original Issue

CONTENTS

Purpose.....	1
Overview	1
Specification and Scope of Implementation	1
Modbus Optional Modes and Communication Parameters.....	1
Function Codes and Maximum Number of Parameters per Query / Response	2
Error Checking, Exception Responses and Communication Failures.....	2
Modbus Slave Implementation	2
Modbus Address Mapping.....	2
Creating the Configuration File.....	3
Configuration Syntax.....	3
Configuration Statements.....	3

PURPOSE

The purpose of this document is to assist users of DRMCC to set up Modbus communications between the UIM acting as Modbus slave and their RTU or other device acting as Modbus master. It is possible to set up a second Modbus slave port and / or a Modbus master port – these will be configured at extra cost by the factory when needed. The document includes a specification and scope of the UIM Modbus implementation with instructions for Modbus slave implementation including configuration and data mapping.

OVERVIEW

The DRMCC includes a “Modbus mapping agent (MMA)” which allows it to act as a device on a Modbus network. Modbus is a widely used industrial control protocol originally developed by Modicon Inc.

MMA is designed to act as an intermediary between the Modbus network and the DRMCC monitoring application. Before MMA can be used the interface between it and the application must be recorded in a configuration file.

SPECIFICATION AND SCOPE OF IMPLEMENTATION

The Modbus implementation conforms to the specification for Modbus Protocol chapters 1 to 6 that may currently be found at the Modicon web site, <http://www.modicon.com/techpubs> , with the following scope:

Modbus Optional Modes and Communication Parameters

Modbus Master and Modbus Slave protocols are supported, only one per serial port. One and only one active Modbus Master shall be connected to each Modbus network. The master polls the slaves with a query, and the slaves send a response.

Modbus ASCII and Modbus RTU protocols are supported, only one per serial port. The serial transmission mode and serial communication parameters shall be the same for all devices connected to a specific Modbus port, but may be different for different ports.



Dynamic Rating, Monitoring, Control and Communications

Modbus User Guide

Function Codes and Maximum Number of Parameters per Query / Response

The following table specifies the Modbus function codes supported and the maximum number of data items that shall be requested or sent in a master query, or returned in a slave response. (However, specific slave devices may have lower limits.) Note that in Modbus, a "word" is 16 bits.

Code	Name	Type	Modbus Address Range	Maximum items
01	Read Coil (Output) Status	Bit	0001 - 9999	2000 coils
02	Read Input Status	Bit	10001 - 29999	2000 inputs
03	Read Holding Registers	Word	40001 - 49999	125 registers
04	Read Input Registers	Word	30001 - 39999	125 registers
05	Force Single Coil	Bit	0001 - 9999	1 coil
06	Preset Single Register	Bit	40001 - 49999	1 register
08	Loop back Test (sub-function 00)	-	-	-
15	Force Multiple Coils	Bits	0001 - 9999	2000 coils
16	Preset Multiple Registers	Words	40001 - 49999	125 registers

Data types supported shall be B (1 bit), W (word = 16 bits), D (double word = 32 bits). A word can be used to hold a Linux short integer. A double word can be used to hold a Linux integer or IEEE 754 floating point number.

Error Checking, Exception Responses and Communication Failures

Error checking capability in Modbus includes parity, LRC, CRC checking.

If the Modbus slave receives a query that it cannot handle, it shall return an exception response in accordance with the Modbus Specification Chapter 4. If the Modbus master receives an exception response, it shall return it to the application program for appropriate action. The exception response includes slave address, function code and exception code.

If the Modbus slave does not receive a query, or detects a communication error (parity, LRC or CRC) it shall not return a response. If the Modbus master does not receive a response it shall process a timeout condition for the query. After a pre-set number of retries, Modbus shall return to the application program the line (port) number that has failed, the slave (device) address and the function code of the failed query.

MODBUS SLAVE IMPLEMENTATION

Modbus Address Mapping

This section specifies how to map internal variables ("DR Names") to Modbus addresses. Please consult the "Modbus Map" page of the Excel "Setup" spreadsheet for default Modbus Address Mapping. The mapping is for a DRMCC controlling one transformer. The spreadsheet shows all mapped variables in the standard DRMCC - this includes all operational data. Some data for maintenance purposes (TC counters etc) is not shown.

The Modbus map is configurable to suit each application, using the spreadsheet. What this means is that DR Names may be selected from the list and mapped to different Modbus addresses subject to:

Each Modbus Address may appear once only

Each DR Name used must be mapped to a Modbus address in the appropriate range:

- 0001 to 9,999 Output status - bit - read / write
- 10,001 to 29,999 Input status - bit - read only
- 30,001 to 39,999 Input register - word - read only
- 40,001 to 49,999 Register - word - read / write

Also, each 32-bit variable must occupy two adjacent words used only for that purpose.



Dynamic Rating, Monitoring, Control and Communications

Modbus User Guide

However a 32-bit floating point number may be mapped to a 16-bit Modbus integer with scaling. It then requires only one word in the Modbus map.

Alarm bits are read-only. Any alarm in a group activates the appropriate LED & relay. The standard mapping shows AL and TR variables mapped to the 30,001+ range. These words use the three least significant bits for alarm or trip status (LSB), alarm acknowledged and alarm enabled. If only the alarm status is required they may be mapped to the 10,001+ range.

It is recommended that all DRMCCs connected to a SCADA system should use the same mapping to simplify the SCADA software. Each DRMCC on a network must have a different Modbus slave device address. The remote Modbus Master must map its internal registers to a slave address and data address.

Creating the Configuration File

The configuration file is a plain text file called "modbus.txt". It is automatically generated using the Setup.xls spreadsheet as part of the set-up procedure described in "DRMCC UIM operations". Instructions for doing this are included in the spreadsheet.

The file may be further edited using any text editor capable of producing "flat" ASCII text files. This should be done only if absolutely necessary and with caution. Note that errors in the text file may cause incorrect operation of the DRMCC. Note also that if the spreadsheet is subsequently used again to generate the config file, this will overwrite the existing file and all changes made in the text editor will be lost.

Configuration Syntax

The configuration file consists of lines, each of which may be

- empty
- a comment line starting with #
- a configuration statement

Configuration statements consist of a 3-character keyword, followed by one or more values. For example the variable map is declared thus:

```
MAP /etc/mbserv/wilsonvars.so
```

and a serial port is declared thus:

```
SLN S1 /dev/ttyS1 19200 802 RTU SLAVE 1
```

The number and format of values is strictly defined.

The order of statements in the configuration file is important. You must declare things before you use them. In particular, the variable map must be declared before you define any shared memory segments or register mappings. Shared memory needs the size of the variable map before it can be attached, and register mappings need the names of the variables.

Configuration Statements

CFG keyword value

sets the internal configuration variable keyword to value. declares the shared-object file containing the variable map for this configuration.

Keyword	Meaning
master_timeout	The timeout between polls of slaves when acting as a master.
retry_timeout	The timeout between retries of messages to slaves when acting as a master. This value must not be greater than master_timeout.
client_name	The file name of the socket used by an application to force setting requests. By changing this name, multiple instances of mbserv can run on a server.



Dynamic Rating, Monitoring, Control and Communications

Modbus User Guide

example:
CFG master_timeout 5

MAP *filename*

declares the shared-object file containing the variable map for this configuration.

example:
MAP /etc/mbserv/varlist.so

SET *filename*

declares that filename should be used to derive an IPC key for the message queue used to notify an application that a write request has been received. It has the secondary effect of enabling such notification.

example:
SET /tmp/msgq

SLN *ident device speed framing protocol role address*

declares a serial port on which to listen for or send Modbus messages. The values have the following meanings:

Name	Meaning
ident	a simple code to identify this serial port, e.g. S1
device	the full pathname of the serial device, e.g. /dev/ttyS1
speed	the bits-per-second rate of the line: must be either 9600 or 19200
framing	the number of databits, the parity and the number of stop bits and the line configuration; this is encoded as a four-character string: <ul style="list-style-type: none">•the first character is the number of data bits (usually 8)•the second is the parity (N = none, O = odd, E = even)•the third is the number of stop bits (1 or 2)•the fourth is the line configuration (T = 2-wire, F = 4-wire).
protocol	the Modbus protocol, either RTU or ASCII
role	the Modbus role performed on this line (MASTER or SLAVE)
address	for SLAVE ports, the Modbus address (number >= 1)

It is possible to declare serial ports which share the same-named serial device. In this case the open device is shared, and a reference count is maintained by mbserv. The framing attributes found in the first SLN statement which names a device are those actually used; attributes in subsequent SLN statements using the same device are ignored.

This feature is used where an mbserv instance must act as both master and slave on the same wire (for example in a master-follower implementation).

example:
SLN S1 /dev/ttyS1 19200 802 RTU SLAVE 1

MEM *ident path rkey wkey*

declares a shared memory segment for use in application/mapper communication. The values have the following meanings:



Dynamic Rating, Monitoring, Control and Communications

Modbus User Guide

Name	Meaning
ident	a simple code to identify this memory segment, e.g. T1
device	the full pathname of the reference file, e.g. /tmp/shm1
rkey	a single-character key used to identify a segment to be accessed by read requests
wkey	a single-character key used to identify a segment to be accessed by write requests

The separate read and write keys are used where controlling applications maintain separate memory areas for calculated results

example:

```
MEM T1 /tmp/shm1 1 A
```

VAR varname mb-register mem-area serial-line slave-addr scaling

declares a mapping between a variable and a Modbus scalar register number.

Because some SCADA systems cannot handle floating-point or 4-byte integers, you can specify a scaling operation to be performed whenever the configured register is read or written. Scaling only applies when mbserv is acting as a slave. Scaling can convert between floating point and integer, optionally applying a multiplier and optional offset to the raw value. On writing the operation is reversed. Note that if a Modbus master reads a scaled value and then writes the same value back, the raw data may not have the same value, due to rounding tolerances and reduction in precision.

When you specify a multiplier and offset, the result is:

$$\text{scaled-value} = (\text{raw-value} * \text{multiplier}) + \text{offset}$$

for reads and

$$\text{raw-value} = (\text{scaled-value} - \text{offset}) / \text{multiplier}$$

for writes.

You must always provide a scaling value: the special character '*' indicates that no scaling is to be applied.

The operands have the following meanings:

Name	Meaning
varname	a variable named in the variable map file
mb-register	a valid Modbus register number capable of holding a scalar value; if the variable is larger than 2 bytes, implicit register mappings are declared for the additional bytes; for a Modbus MASTER this is the register address on the remote slave device which corresponds to the memory location
mem-area	the identifier of the shared memory segment holding the variable; if this value is '*' and the variable is used in MASTER mode, then the variable is excluded from any polling of slaves, but will be used when an application issues 'set' requests
serial-line	the identifier (as in an SLN statement) of the serial line on which requests for this register will come or on which a Modbus MASTER will send requests
slave-addr	the slave address to which a Modbus MASTER will send requests; the field is ignored (but must be present) for a SLAVE
scaling	a scaling and conversion indicator used when in SLAVE mode: * = no scaling or conversion $Sx = (\text{read})$ multiply the actual value by x , and round to a short integer



Dynamic Rating, Monitoring, Control and Communications

Modbus User Guide

Name	Meaning
	Sx = (write) convert 2 message bytes to a float, then divide by x Lx = (read) multiply the actual value by x , and round to a long integer Lx = (write) convert 4 message bytes to a float, then divide by x x denotes any <i>real</i> number, e.g. 3 or 4.56, with an optional <i>integer</i> offset, separated by a + or - sign. There must be no spaces in the scaling string.

example:

```
VAR sval1 40001 T1 S1 1 S10+3
```

BIT *varname bitpos mb-register mem-area serial-line slave-addr*

declares a mapping between a variable and a Modbus single-bit register number.

Obviously, no scaling operation is provided.

The values have the following meanings:

Name	Meaning
varname	a variable named in the variable map file
bitpos	the bit in the variable; 0 is the least significant and 15 is the most significant; the maximum bitpos value depends on the size of the variable - for char it is 7, for short, 15 and for int, 31
mb-register	a valid Modbus register number capable of holding a scalar value; if the variable is larger than 2 bytes, implicit register mappings are declared for the additional bytes; for a Modbus MASTER this is the register address on the remote slave device which corresponds to the memory location
mem-area	the identifier of the shared memory segment holding the variable; if this value is '*' and the variable is used in MASTER mode, then the variable is excluded from any polling of slaves, but will be used when an application issues 'set' requests
serial-line	the identifier (as in an SLN statement) of the serial line on which requests for this register will come or on which a Modbus MASTER will send requests
slave-addr	the slave address to which a Modbus MASTER will send requests; the field is ignored (but must be present) for a SLAVE

example:

```
BIT bval1 4 101 T1 S2 3
```